Co-funded by the European Union

AI PROJECTS

Technical school Pirot

KA220-VET - Cooperation partnerships in vocational education and training

Project Title: AI tools for VET schools

Document Date: March 2026
This material has been compiled and prepared for purposes of Erasmus project by:
Technical school Pirot (author: Bojan Ćirić, co-authors:Boban Blagojević, Aleksandar Madić)
ICEP (author: Ladislav Mariš, co-author: Adelaida Fanfarova)
Agrupamento de Escolas Tomas Cabreira (author: Sandra Nobre, co-authors: Rui Dias, Gilherme Mota, Carla Lima)
Translated by: Bojana Stojanović

ADDRESS INFORMATION

Takovska 22, Pirot, Serbia
Web: https://book.tsp.edu.rs

Contents

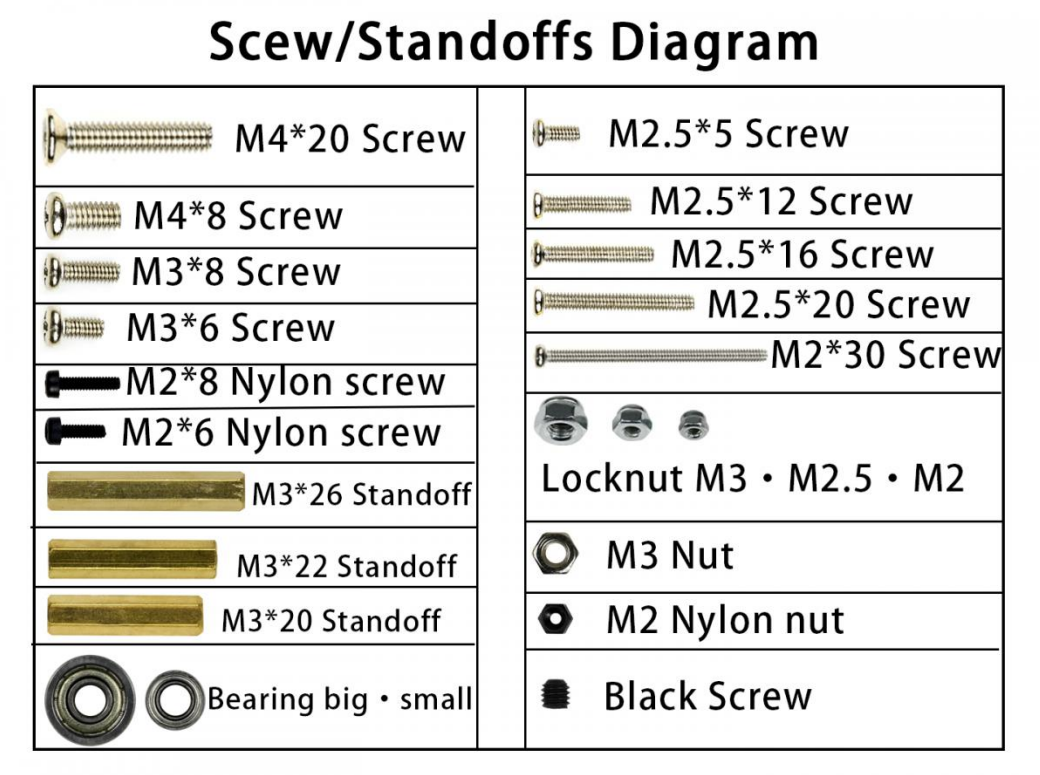
I. PiRACER AI AUTOPILOT

1. PI-RACER DEEP LEARNING AUTOPILOT

1.1 PiRacer Assembly Manual

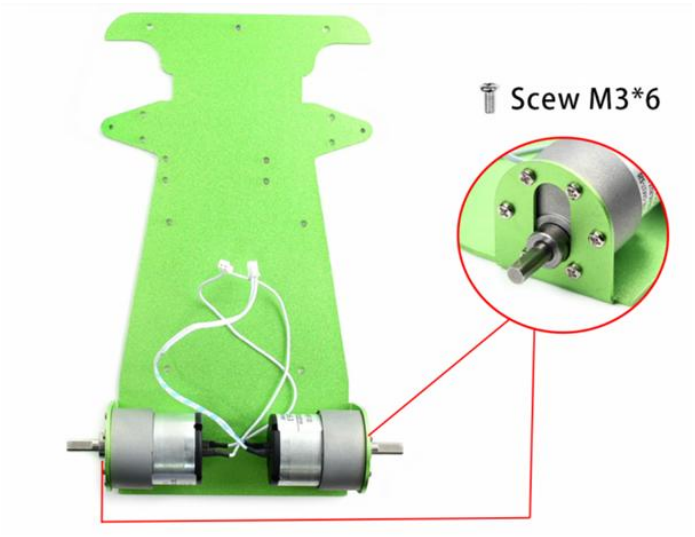
Screws/standoffs diagram

Diagram for reference. Note that the screws that come with servo wheel are not listed here.



1. Attach the motors to metal chassis

Note: Do not use the M3*8. It is longer and might damage the motor.



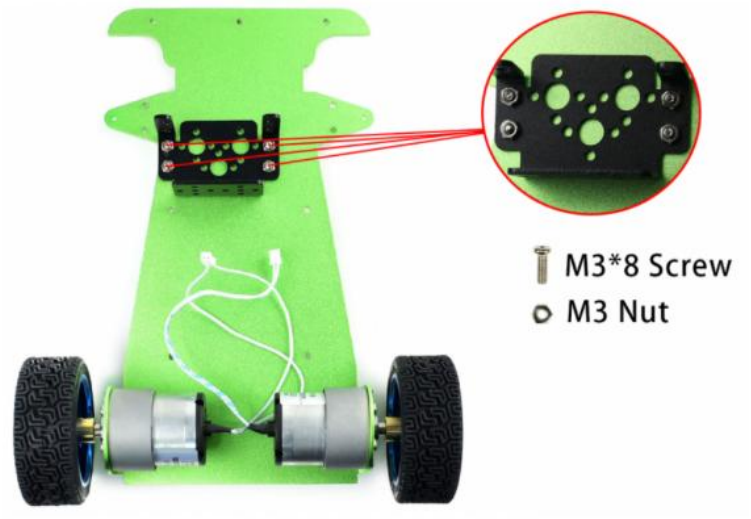
2. Add couplers to wheels

First, insert the black screw into the coupler. Then add the coupler to the wheel. You may need to press the coupler into the wheel. Secure the coupler to the wheel using M4*8 screw.

3. Assemble the wheels

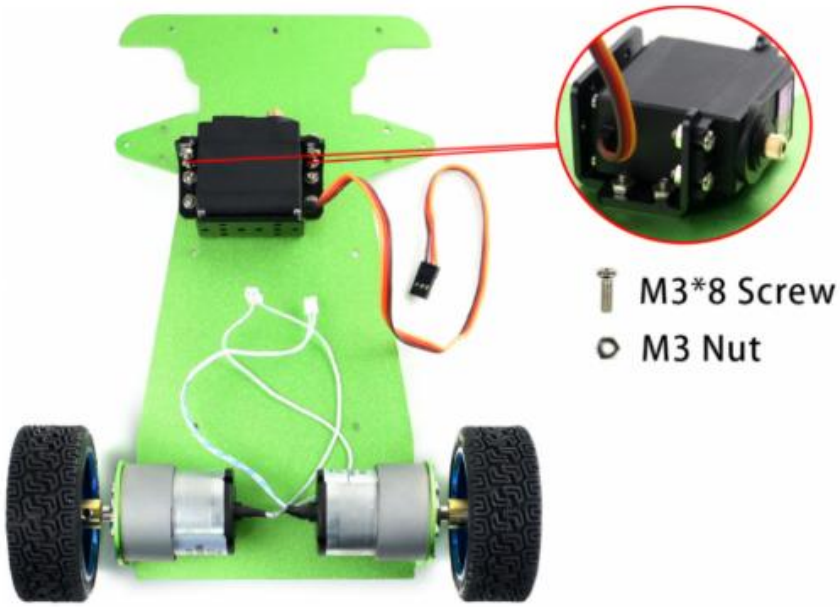
Tighten the black screw to secure the coupler to the flat side of the axel

4. Mount the servo holder on metal chassis



5. Attach the servo on the holder and using the screws and nuts

Make sure the servo is in correctly. The outer shaft should be in the center.

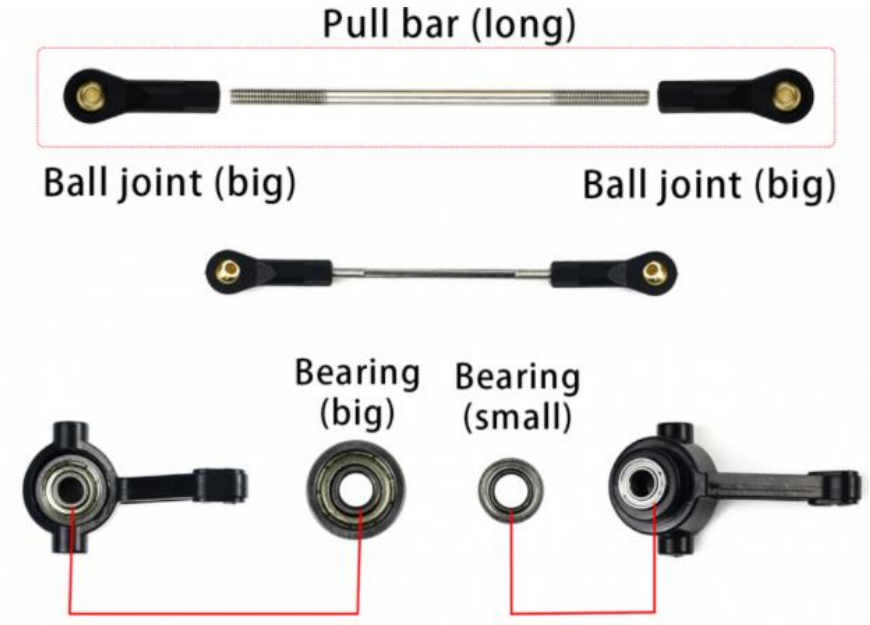


6. Assemble the servo pull bar

The pull bar is combined by using two ball joints, one flat and one ball, and the short bar. The two ball joints should be perpendicular to each other. Attach the servo wheel (horn) to the servo wheel holder using the screws that came with the servo wheel. Then attach flat ball joint to the servo wheel holder. Note that the groove of the servo wheel is toward outside.

7. Assemble the front-wheel pull bar and the steering knuckles

The front-wheel pull bar is combined by using two ball joints and the long bar. Then put the bearings in the steering knuckles. Each knuckle needs a small and a large bearing.



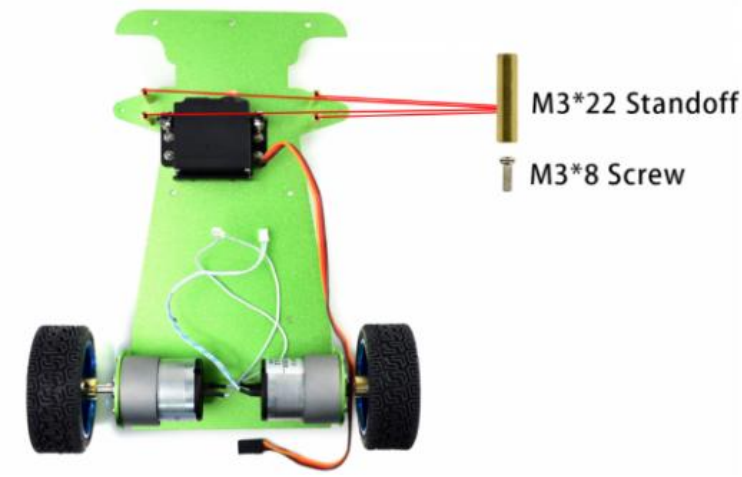
8. Assemble the servo pull bard, front-wheel pull bard, and the steering knuckles

Attach the servo pull bar on the top, and then the front-wheel pull bar, and finally the knuckles. The larger bearing should towards the inside.

9. Attach the wheels on the steering knuckle

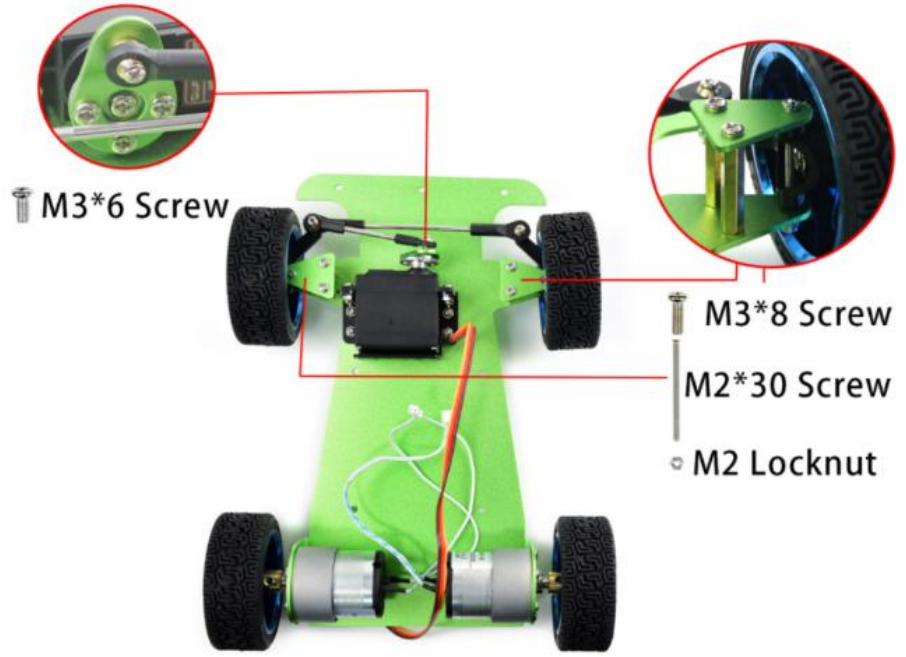
The nut should be on the outer side of the wheel, opposite the knuckle. Make sure the wheel is not too tight or too loose. Test the wheel to make sure it can move freely.

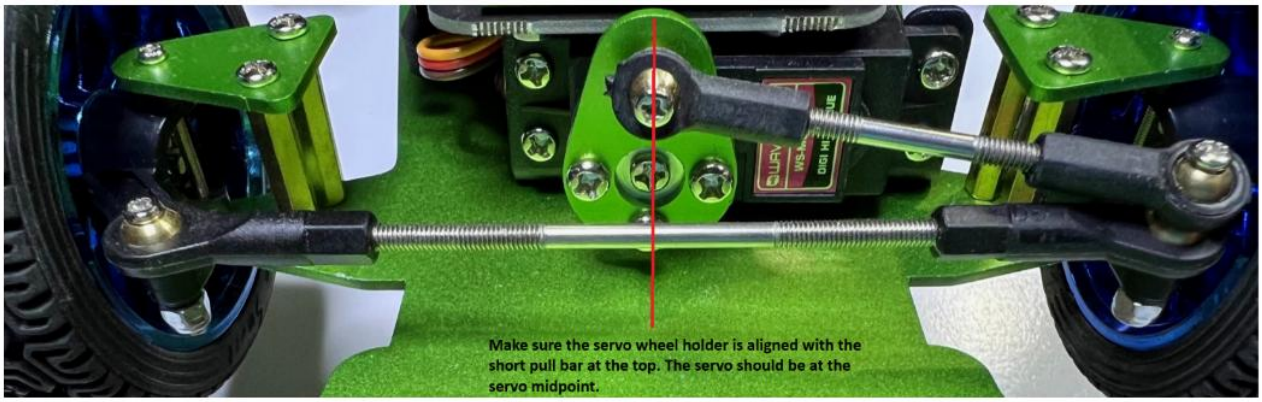
10. Add the M3 standoffs for the front wheels



11. Assemble the front-wheels combination

Put the servo wheel to servo, fix it by M3 screw. Fix the wheels by M2 screws and locknut and the triangle board. The front wheels should be straight forward. Adjust the long pull bar if needed.

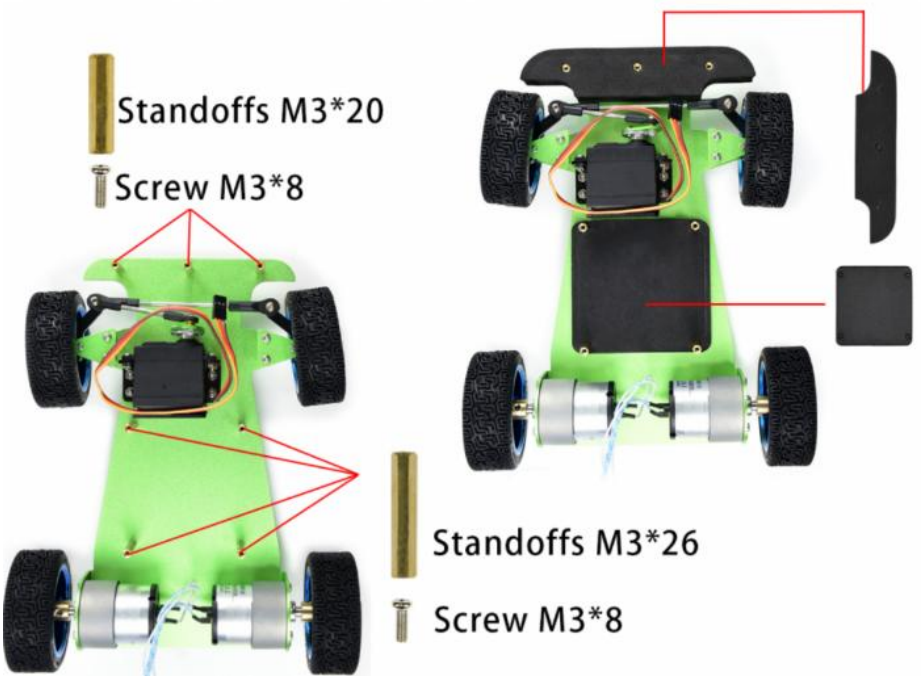




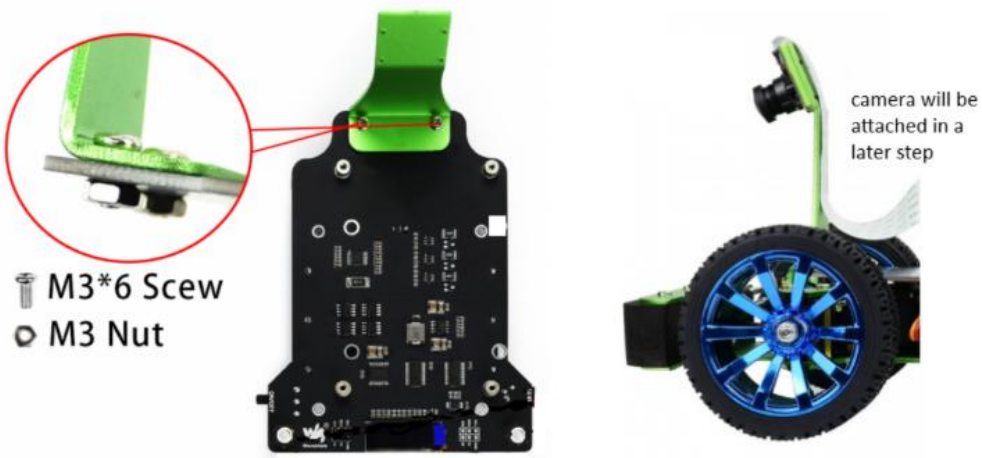
The front wheels should be straight forward. Adjust the long pull bar if needed.

12. Add the standoffs for PiRacer Expansion board and the bumper

Insert the EVA felt pads for the bumper and the PiRacer Expansion board.

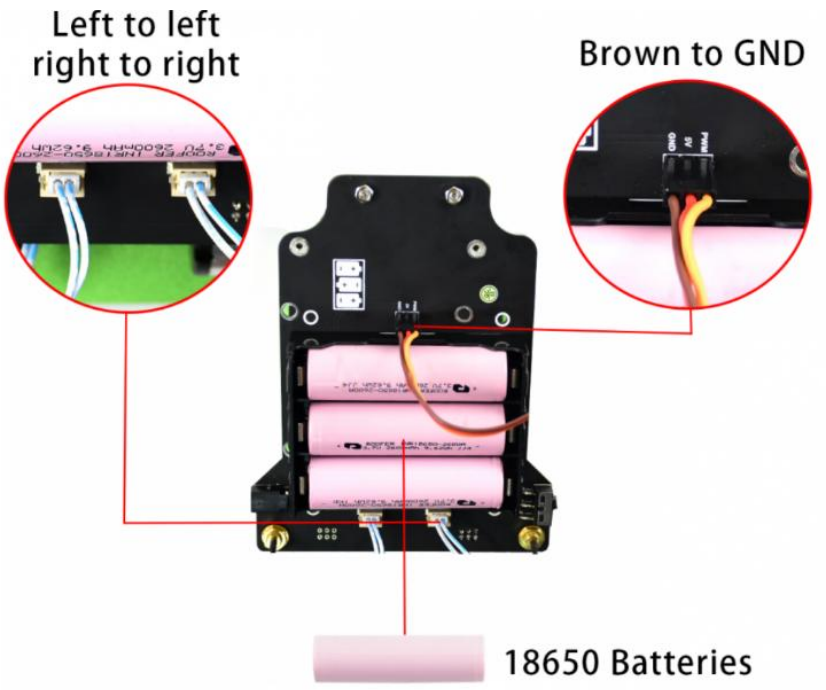


13. Attach the camera holder to PiRacer Expansion board.



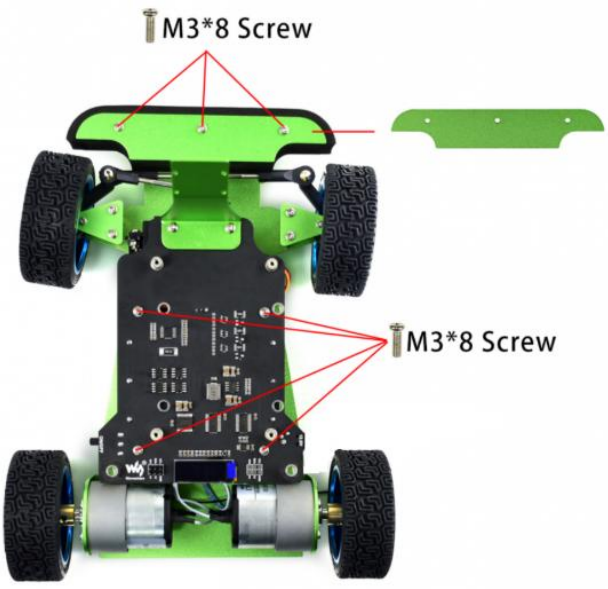
14. Insert the batteries in the correct direction

Connect the wires of the motors and servo to PiRacer Expansion board.



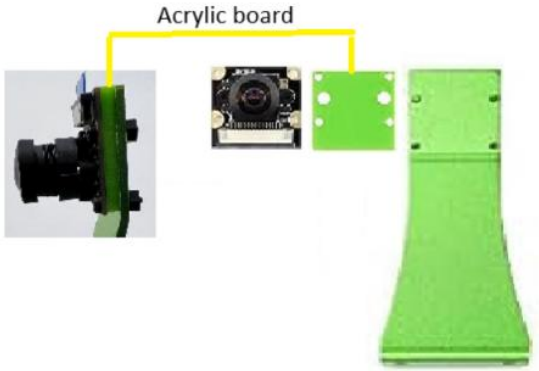
15. Attach the Expansion board and the metal bumper

Adjust the placement of motor and servo wires and attach the PiRacer Expansion board to the metal chassis and attach the metal bumper.

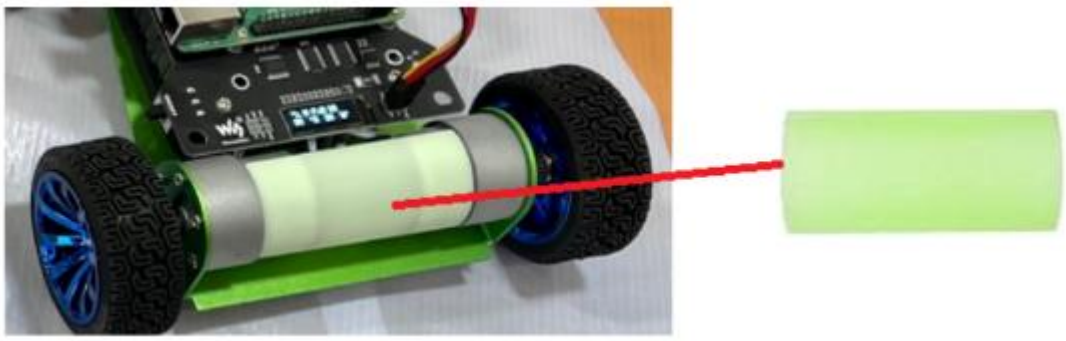


16. Mount camera to its holder using the nylon screws

Note: The Acrylic board should be between camera and the metal holder to avoid shorting.



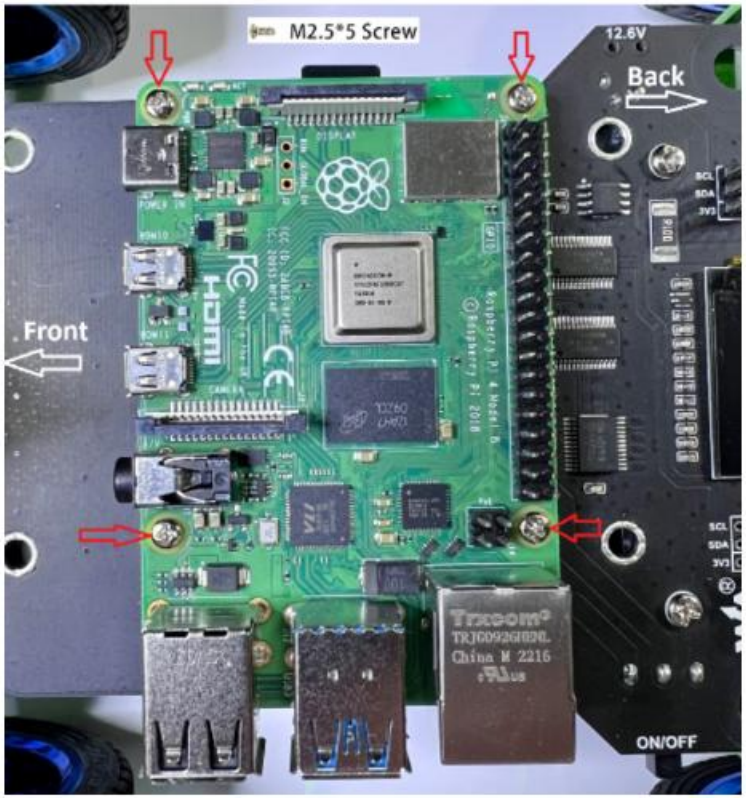
17. Attach the 3D-printed motor enclosure on DC motors



18. Power off the Raspberry Pi and disconnect the charger from the PiRacer Expansion board before proceeding.

19. Attach the Raspberry Pi to the PiRacer Expansion board

GPIO pins should be at the back of the car.

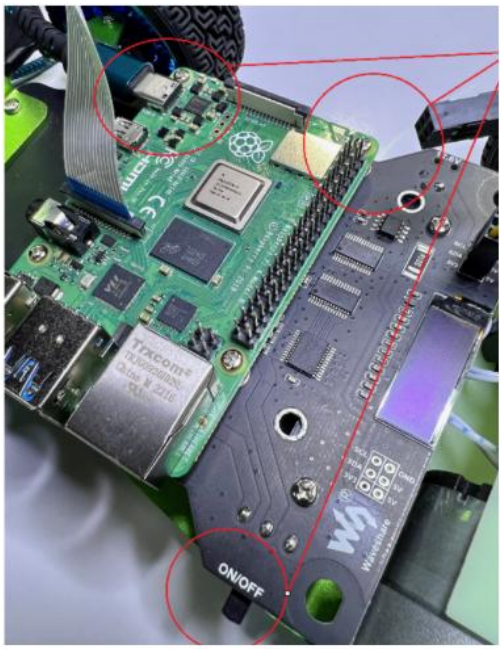


21. Attach the camera ribbon cable to the Raspberry Pi camera input

Blue side towards the Pi's USB ports.

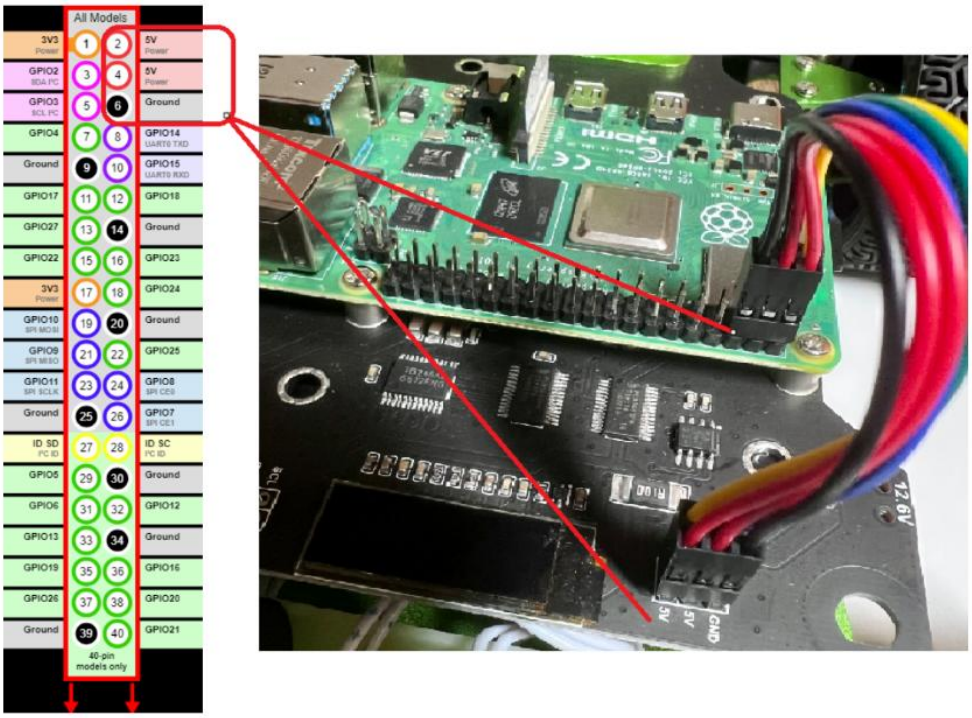
****WARNING ** Make sure the Raspberry Pi is not powered when connecting the 6 pin wires**

Also do not power the Raspberry Pi through the USB-C (external power) while it is powered by the PiRacer Expansion board.



22. Connect the Raspberry Pi to the PiRacer Expansion board using the 6PIN wires

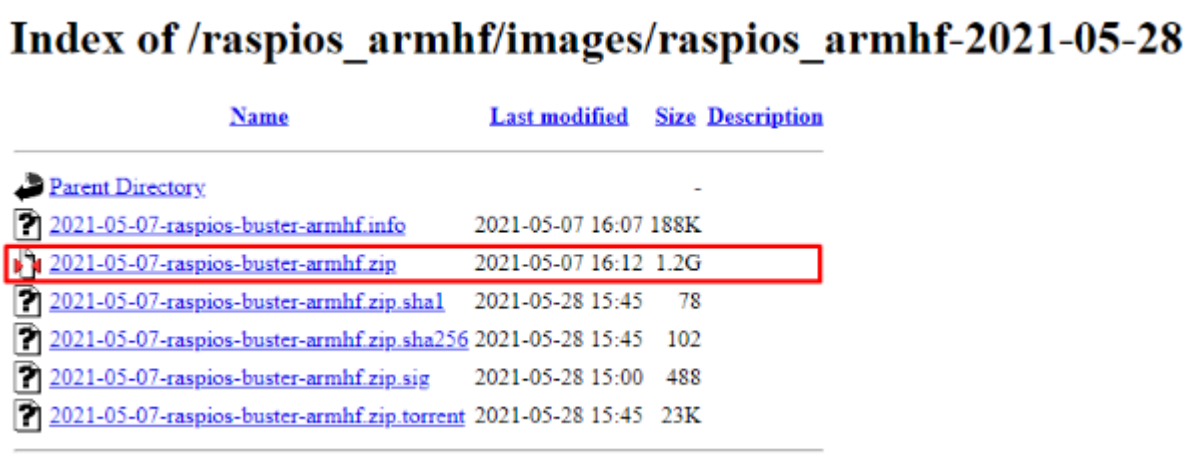
Match the red|red|black (5V|5V|ground) on the Pi GPIO and the black|red|red (gnd|5V|5V) on the PiRacer Expansion board.



1.2 Raspian Legacy (Buster) Desktop

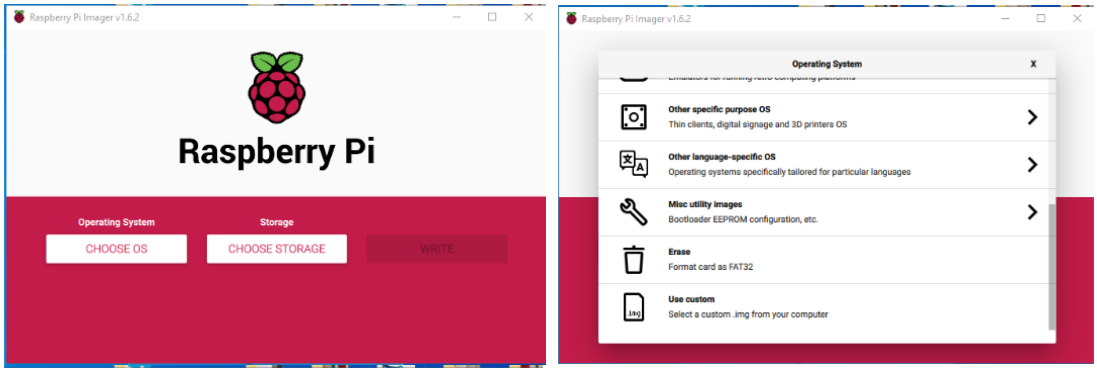
Flash OS - Raspian Legacy (Buster) Desktop

Navigate to the official download repository for Buster OS. **All earlier versions of Raspberry Pi OS can be found and downloaded here** and the directly previous Raspberry Pi 'Buster' OS **official download link is here**. See below for what that looks like, download the image file by clicking on the highlighted file.

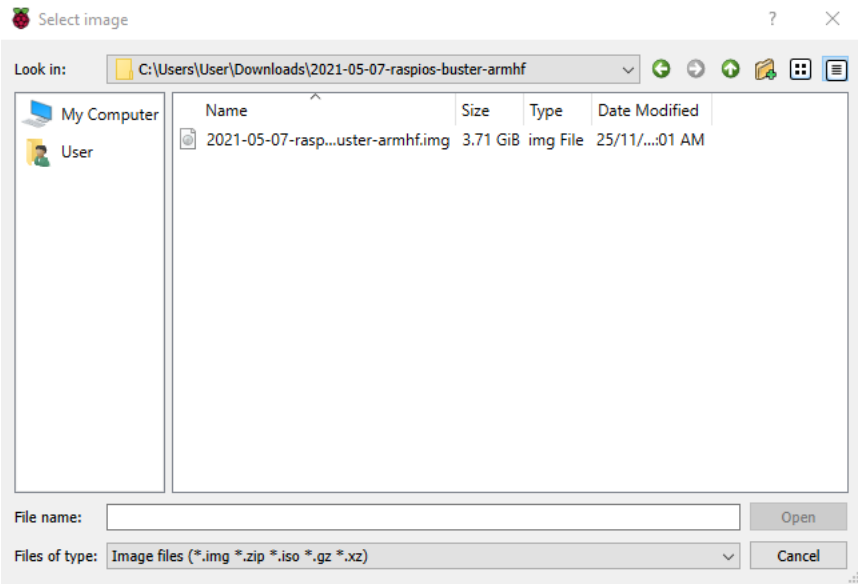


Download and Install Raspberry Pi Imager from https://www.raspberrypi.com/software/

Now, let's open up the Official Raspberry Pi Imager, see it in the image below. Worth noting here - if you hit **| CTRL+SHIFT+X |** on your keyboard while in the Raspberry Pi Imager Program it will open the Advanced Hidden Menu. This Hidden Menu allows you to preconfigure your Raspberry Pi with SSH, WIFI credentials, and Localisation settings.



Having done this it will open up a file explorer. Navigate to that extracted Disc Image File and click on it. See this in the image below. Then open the file image and this will arm the Raspberry Pi Imager with the Raspberry Pi 'Buster' OS. Find it, select it, and open it.



Insert the Micro-SD card that you want to be flashed into your computer. Use a **USB to Micro-SD adapter** if needed. Then click on the **| CHOOSE STORAGE |** button and select your inserted Micro-SD. Keep in mind any data that was on your Micro-SD card will be wiped/permanently deleted when it is flashed. The Official Raspberry Pi Imager will now look like the image below.

With everything sorted (the right OS loaded and the correct Storage selected) you can now click the **| WRITE |** button to start the flashing process. See this flashing process in the image below.

Once the flash is complete it will automatically virtually eject the Micro-SD card from the computer. So then you can simply physically take out your Micro-SD and insert it into a Raspberry Pi single-board computer. Then set your Raspberry Pi up normally as a desktop computer. Once it boots you will be greeted by the old familiar background, see image below, and you will have successfully flashed 'Buster' OS to your Raspberry Pi. You are free now to roam your well-acquainted digital ground.

Follow Pi Wizard instruction to set location, keyboard, wifi, and get update

Menu / Preferences / Raspberry Pi Configuration/ Interfaces

- enable Camera and I2C
- optional: enable VNC for remote access
- Click OK and reboot

Additional Software

Java 3.8.3 (run one at a time)

```
sudo apt install build-essential libncurses5-dev libgdbm-dev libnss3-dev libssl-dev
libreadline-dev libffi-dev -y
wget https://www.python.org/ftp/python/3.8.3/Python-3.8.3.tgz
tar -zxvf Python-3.8.3.tgz
cd Python-3.8.3
sudo ./configure --enable-optimizations
sudo make -j 4
sudo make altinstall
sudo python3.8 -m pip install boto3
sudo python3.8 -m pip install tqdm
```

Additional tools

```
sudo apt install chromium-browser -y
sudo apt-get install zip unzip -y
```

AWS CLI

Default user account on the Raspberry Pi

```
sudo apt install awscli -y
sudo pip3 install --upgrade awscli
sudo pip3 install boto3

user - pi
pw - raspberry
```

1.3 WaveShare Branch for DonkeyCar Software

Install Dependencies

```
sudo apt-get install build-essential python3 python3-dev python3-pip
python3-virtualenv python3-numpy python3-picamera python3-pandas
python3-rpi.gpio i2c-tools avahi-utils joystick libopenjp2-7-dev
libtiff5-dev gfortran libatlas-base-dev libopenblas-dev
libhdf5-serial-dev git ntp -y
```

Optional?

```
sudo apt-get install libilmbase-dev libopenexr-dev libgstreamer1.0-dev
libjasper-dev libwebp-dev libatlas-base-dev libavcodec-dev libavformat-dev
libswscale-dev libqtgui4 libqt4-test -y
```

Setup Virtual Environment

```
python3 -m virtualenv -p python3 env --system-site-packages
echo "source ~/env/bin/activate" >> ~/.bashrc
source ~/.bashrc
```

Install DonkeyCar Python Code - WaveShare Branch for DonkeyCar Software 3.1.0

```
mkdir projects
cd projects
git clone https://github.com/waveshare/donkeycar

cd donkeycar
git checkout master
pip install -e .[pi]

pip install tensorflow==1.13.1
```

Not needed pip install numpy --upgrade

```
pip install protobuf==3.20.*
```

Test tensorflow version - should show 1.13.1

```
python -c "import tensorflow; print(tensorflow.__version__)"
```

NOTE: Could not run model on car until running the following two

```
pip install https://github.com/lhelontra/tensorflow-on-arm/releases/download/v2.2.0/
        tensorflow-2.2.0-cp37-none-linux_armv7l.whl

pip install tensorflow==1.13.1
```

Edit camera.py to add self.camera.hflip = True

```
sudo nano /home/pi/projects/donkeycar/donkeycar/parts/camera.py
```

```
class PiCamera(BaseCamera):
    def __init__(self, image_w=160, image_h=120, image_d=3, framerate=20):
        from picamera.array import PiRGBArray
        from picamera import PiCamera

        resolution = (image_w, image_h)
        # initialize the camera and stream
        self.camera = PiCamera() #PiCamera gets resolution (height, width)
        self.camera.vflip = True
        self.camera.hflip = True
        self.camera.resolution = resolution
        self.camera.framerate = framerate
        self.rawCapture = PiRGBArray(self.camera, size=resolution)
        self.stream = self.camera.capture_continuous(self.rawCapture,
            format="rgb", use_video_port=True)

        # initialize the frame and the variable used to indicate
        # if the thread should be stopped
```

Optional Install OpenCV

```
sudo apt install python3-opencv -y
```

Test with:

python -c "import cv2"

1.4 Getting Started with DonkeyCar

Open a terminal window enter the following command

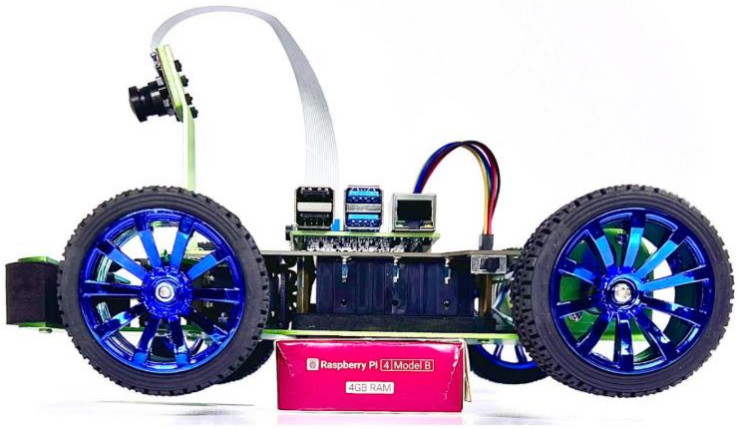
Create DonkeyCar App

This will create a folder called mycar with all the python code needed to drive the car.

Calibrate the front steering

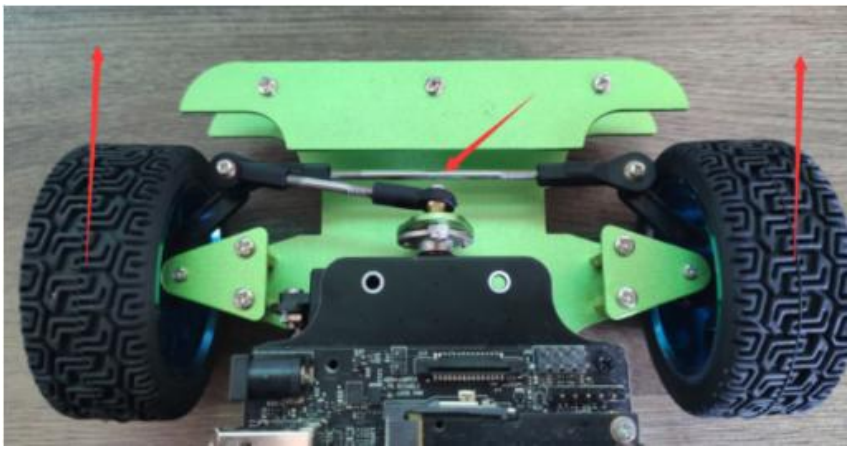
Make sure your car is off the ground to prevent a runaway situation.

Use a small box like the on the Raspberry Pi came in.

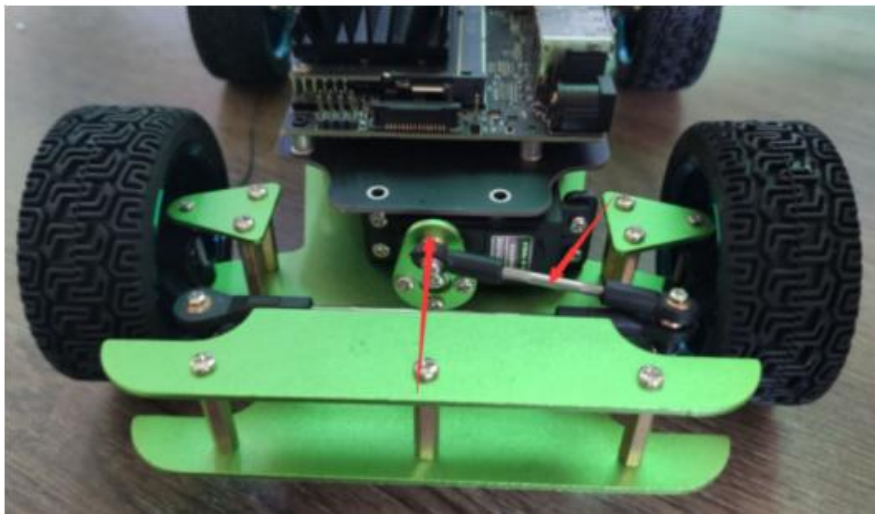


To make sure that the DonkeyCar can drive straight and make the needed turns on the track, the car's hardware and software need to be calibrated.

The front wheels should be straight forward. Adjust the long pull bar if needed



The servo wheel holder should be aligned with the short pull bar at the top. Adjust the short pull bar if needed.



Calibrate the servo software

Find the the left, center and right steering PWM for this car's servo.

The center should be half way between the left and right. example:

Left 200
right 560
center will be 380

In a terminal window enter the following

```
cd ~/mycar
donkey calibrate --channel 0 --bus=1
```

Try values 300, 400, 500 and see how the steering changes. Figure out the max turn for left and right.

Once you have the numbers, edit the config.py and update the values you found.

The throttle should be set using your numbers for the steering. The throttle is already set correctly.

nano config.py

```
#STEERING
STEERING_CHANNEL = 0            #channel on the 9685 pwm board 0-15
STEERING_LEFT_PWM = 200         #pwm value for full left steering
STEERING_RIGHT_PWM = 560        #pwm value for full right steering

#THROTTLE
THROTTLE_CHANNEL = 0            #channel on the 9685 pwm board 0-15
THROTTLE_FORWARD_PWM = 4095     #pwm value for max forward throttle
THROTTLE_STOPPED_PWM = 0        #pwm value for no movement
THROTTLE_REVERSE_PWM = -4095    #pwm value for max reverse throttle
```

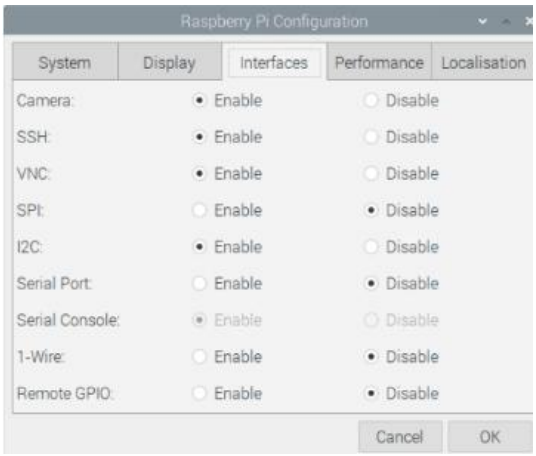
Install OLED Display Service

```
cd ~
git clone https://github.com/waveshare/pi-display
cd pi-display
sudo ./install.sh
cd ~
```

2. RASPBERRY PI REMOTE ACCESS USING REALVNC

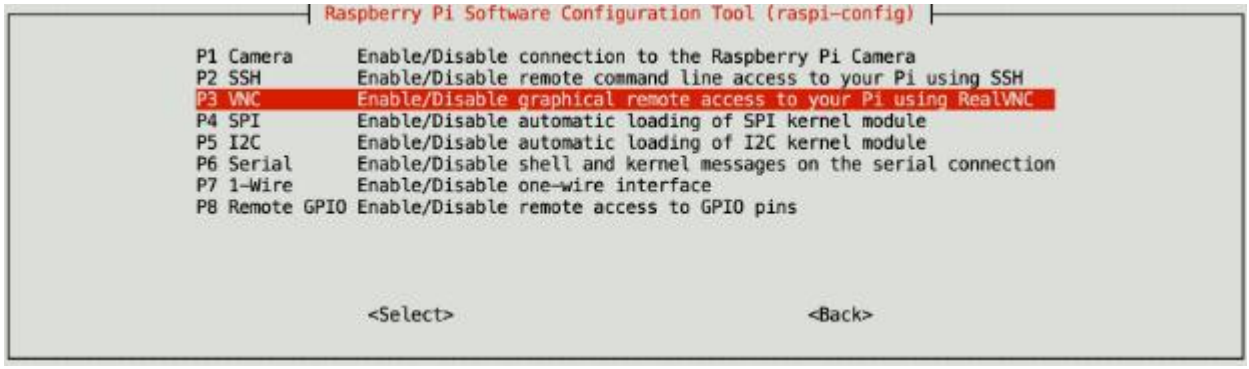
2.1 Enable VNC in Raspian OS with a or b:

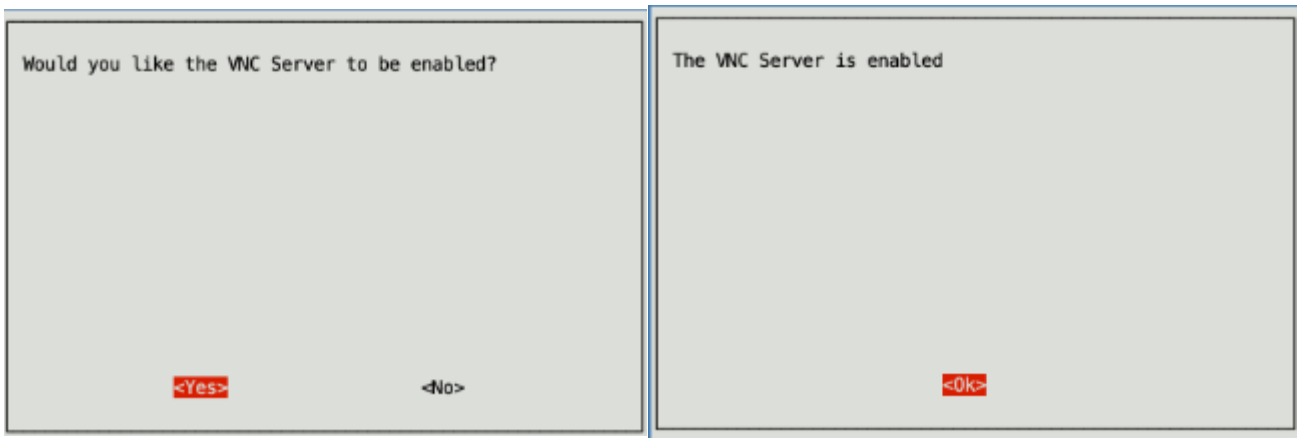
a) Enable VNC in Preferences - Raspberry PI Configuration – Interfaces



b) Another method to enable VNC is using Terminal, enter the command

```
sudo raspi-config
```

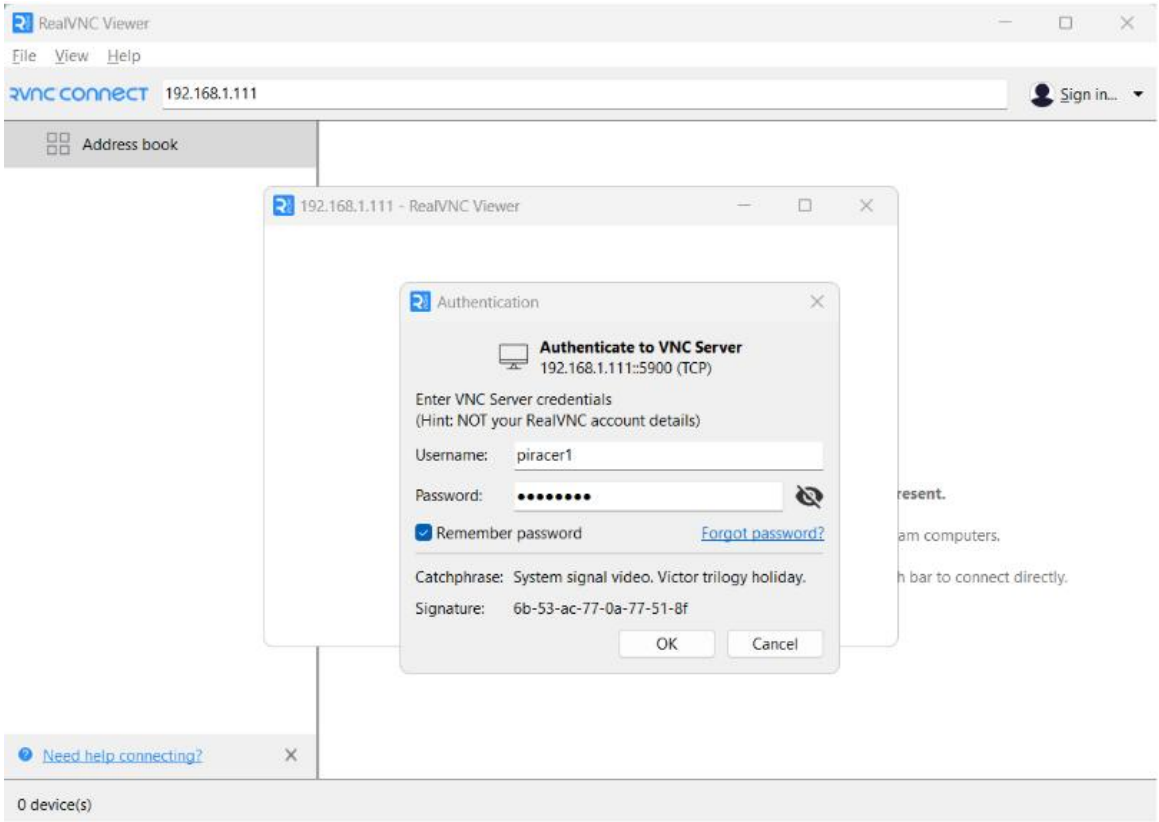




2.2 Install a VNC Viewer

You will need to install a VNC Viewer on your computer, so you can connect to you Raspberry Pi. There are a number of viewers available, but the easiest to set up is **Real VNC Viewer**. You can download Windows and Mac installers from here: https://www.realvnc.com/en/connect/download/viewer/

Open Real VNC Viewer and enter Piracer IP adress (read it from car display)



3. START DRIVING AND COLLECT DATA

Start your car

```
cd ~/mycar
python manage.py drive
```

This script will start the drive loop in your car which includes a part that is a web server for you to control your car. You can now control your car from a web browser at the URL: <your car's hostname.local>:8887

Open a browser and connect to the DonkeyCar Monitor at **localhost:8887**

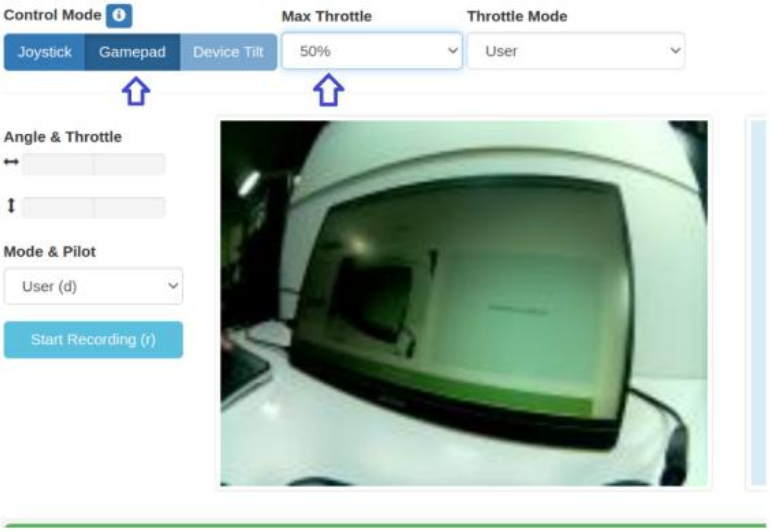
NOTE: When the car is started, a folder created under /mycar/data called tub_#_date to store the data for the session. The data is gathered when the throttle is engaged. To gather a clean dataset, best practice is to stop the "python manage.py drive" using **Contole + C** and restart to begin the training with a new /mycar/data/tub.

There are 2 drive options

1. Use the gamepad (recommended)

After starting the car, open a web browser from the Raspberry Pi desktop while the car is connected to the monitor.

Select Gamepad and set the throttle to 50% to start practicing. Test turns and speed while the car is on the Pi box.

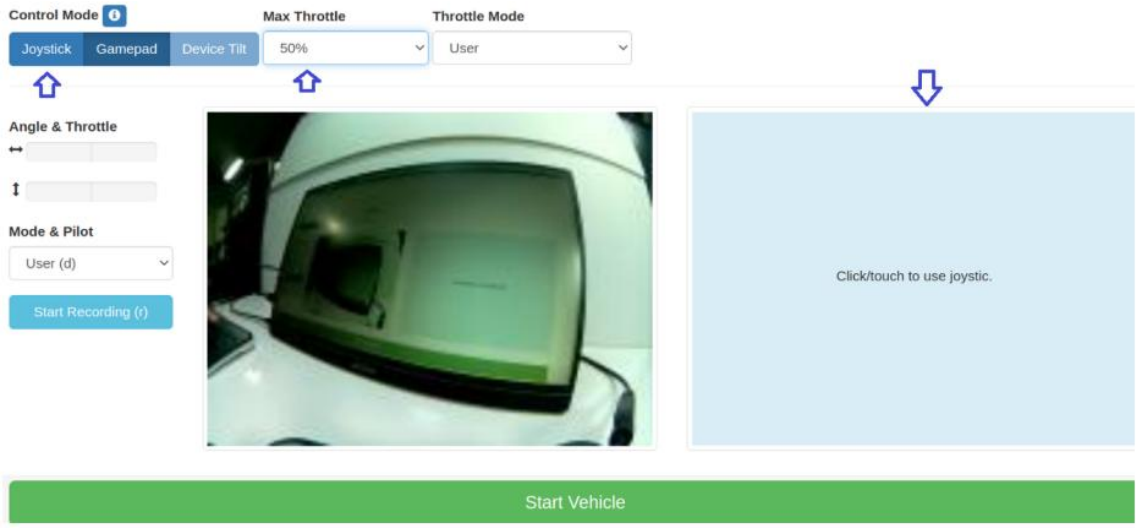


When you are ready to drive on the track, disconnect the monitor, place the car on the track and start driving. Start slow until you are comfortable handling the car.

2. Use the web browser joystick

You will need to use a tablet or phone connected to the same wifi. After starting the car, use a tablet or phone to connect to the car's web server using the car's IP:8887. The car should display its IP if the OLED Display Service was enabled in a previous step.

Select Joystick (the joystick control area is labeled **click/touch to use joystic**). Use the joystick area to drive the car with your finger.



When finished driving, stop the "python manage.py drive" using **Contol + C**.

4. TRAIN DATA - CREATE INFERENCE MODEL

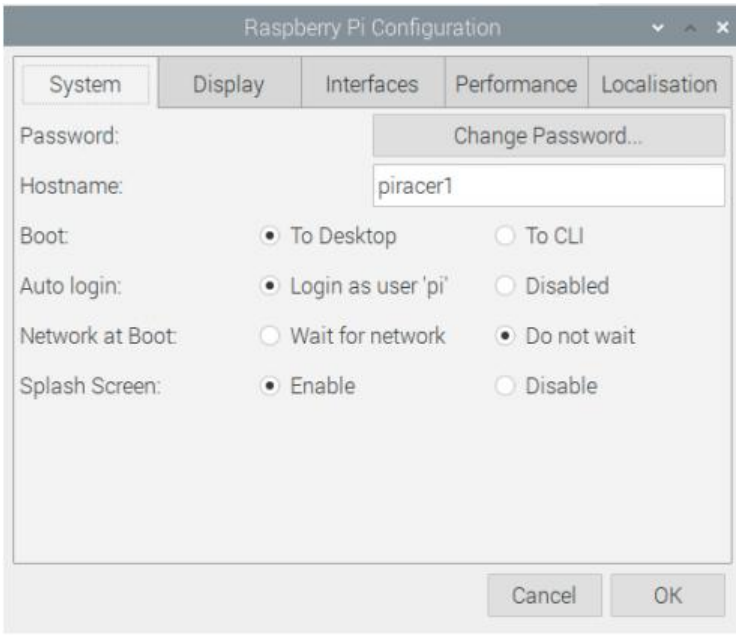
Data location is /home/pi/mycar/data. All tub folders in this directory will be used to create your model. Remove anything you don't want included. You can train data with Raspberry Pi or with AWS virtual machine .

1. Train with Raspberry Pi

Train Data:
In a new terminal session on your host PC use rsync to copy your cars folder from the Raspberry Pi

```
rsync -rv --progress --partial <hostname>@<your_pi_ip_address>:~/mycar/data/
~/mycar/data/
etc:
rsync -rv --progress --partial piracer1@192.168.1.111:~/mycar/data/  ~/mycar/data/
```

You can find hostname in Raspberry Pi Configuration:



Train model:

In the same terminal you can now run the training script on the latest tub by passing the path to that tub as an argument. You can optionally pass path masks, such as ./data/* or ./data/tub_?_17-08-28 to gather multiple tubs. For example:

```
python ~/mycar/manage.py train --tub <tub folder names comma separated> --model
./models/mypilot.h5
etc:
python ~/mycar/manage.py train --tub ./data/tub_1_24-04-06 --model
./models/mypilot.h5
```

It will cost a long time to train a model, please be patient.

After training, you can get a model, you need to copy the module to your raspberry Pi and test it.

```
rsync -rv --show-progress --partial ~/mycar/models/
<hostname>@<your_ip_address>:~/mycar/models/
etc:
rsync -rv --show-progress --partial ~/mycar/models/
piracer1@192.168.1.111:~/mycar/models/
```

5. LOAD MODEL AND DRIVE AUTONOMOUSLY

Start your car with the model

```
cd ~/mycar
python manage.py drive --model ~/mycar/models/mypilot.h5
```

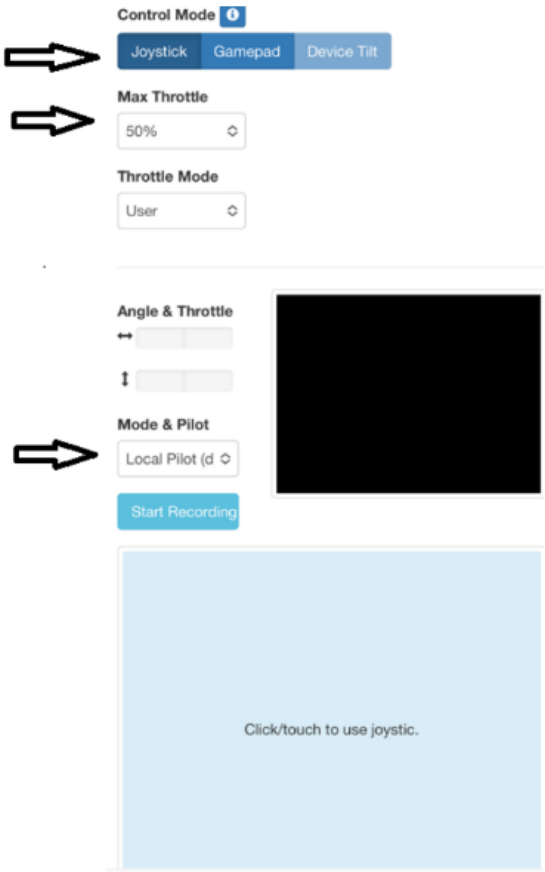
For autonomous driving or steering, use the web browser joystick on another device.

You will need to use a tablet or phone connected to the same wifi. After starting the car, use a tablet or phone to connect to the car's web server using the car's IP:8887. The car should display its IP if the OLED Display Service was enabled in a previous step.

Autonomous Driving

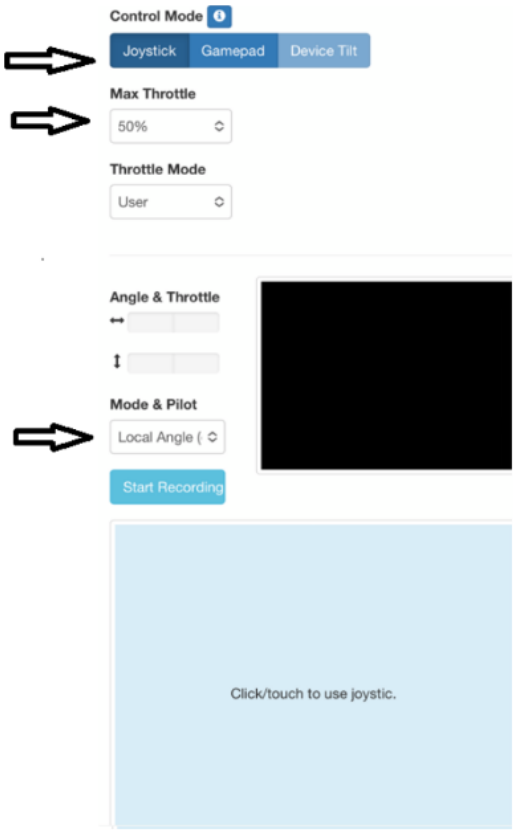
NOTE: As soon as **Local Pilot** is selected the car will start autonomous driving

- **Control Mode:** Select **Joystick ** (the joystick control area is labeled **click/touch to use joystic**)
- **Max Throttle:** Set the Max throttle to about 50% to keep the car from going too fast
- **Mode & Pilot:** Select Local Pilot (Local Pilot is autonomous)



Autonomous Steering

- **Control Mode:** Select **Joystick ** (the joystick control area is labeled **click/touch to use joystic**)
- **Max Throttle:** Set the Max throttle to about 50% to keep the car from going too fast
- **Mode & Pilot:** Select Local Angle (Local Angle is the car steering while you provide the throttle)



Faster autonomous model?

Use your working autonomous driving model to gather new data at a faster speed. Use Local Angle so the car steers while you provide faster throttle.

II. AI WITH AR/VR

6. INTRODUCTION

The Meta Quest 3 marks a pivotal shift from pure Virtual Reality (VR) to high-fidelity Mixed Reality (MR). By leveraging high-resolution color passthrough and a significantly more powerful Snapdragon XR2 Gen 2 chipset, it allows digital content to coexist seamlessly with the physical world. With its slimmer "pancake" optics and 4K+ Infinite Display, it is currently the most accessible powerhouse for both consumers and developers.

6.1 Introduction to Meta Quest 3 and Mixed Reality

The Meta Quest 3 utilizes high-resolution Color Passthrough technology. Unlike traditional VR, where the world is entirely virtual, **Mixed Reality (MR)** uses onboard cameras to project the real environment and then overlays **WebGL** elements on top. Your code utilizes the immersive-ar mode, which is the core of this spatial experience.

System Architecture (The Web Stack)

The application functions as a "sandwich" of layers:

- **Layer 1 (Hardware)**: Quest 3 sensors, depth projectors, and RGB cameras.
- **Layer 2 (Browser)**: Meta Quest Browser (Chromium-based with WebXR support).
- **Layer 3 (API)**: WebXR Device API, which acts as the bridge between hardware and code.
- **Layer 4 (Logic)**: Your JavaScript logic using Three.js for rendering and TensorFlow.js for vision.

7. WEBXR: THE BROWSER-BASED ALTERNATIVE

WebXR allows developers to create immersive experiences that run directly in the Meta Quest Browser. It is built on standard web technologies, making "VR on the web" as easy to access as a website.

Why Choose WebXR?

- Frictionless Access: Users don't need to download large files from the Meta Store; they simply click a URL and hit "Enter VR."
- Frameworks: It utilizes powerful JavaScript libraries like Three.js, A-Frame (HTML-based), or Babylon.js.
- Cross-Platform: A single WebXR app can often run on a Quest 3, an Android phone (AR mode), or a desktop PC.

7.1 Library Analysis

Three.js (The Render Engine)

Three.js handles the heavy lifting of WebGL. It manages the Scene, Camera, and Renderer. In your code, the renderer is set to alpha: true, which is vital for AR as it allows the "empty" parts of the browser to become a window into the real world.

TensorFlow.js & COCO-SSD

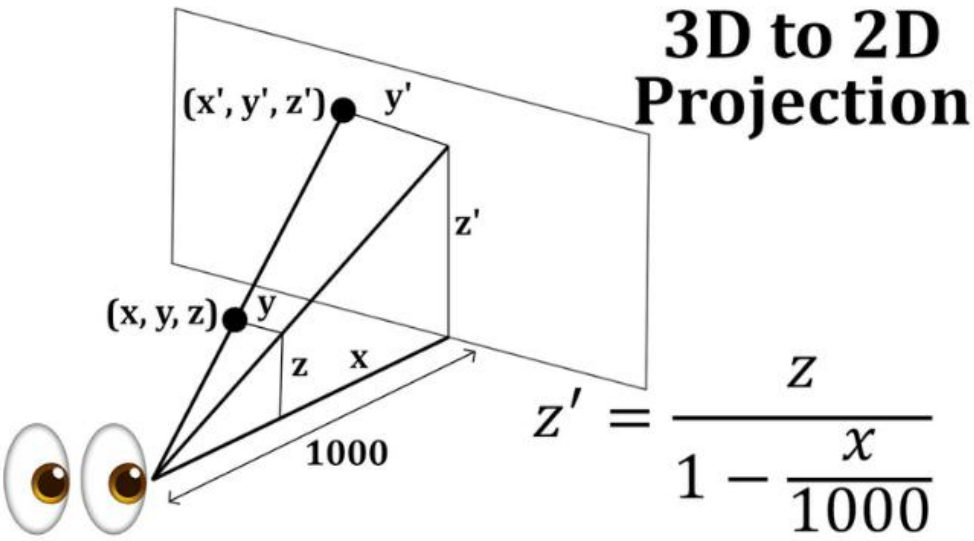
COCO-SSD (Common Objects in Context - Single Shot MultiBox Detector) is a model trained to recognize 80 classes of objects. The beauty of TF.js is its **WebGL backend**, which ensures that AI calculations are performed on the Quest's GPU rather than the CPU, preventing the device from overheating.

7.2 Mathematical Projection (2D to 3D)

This is the most technical part of the script. The AI model returns a bbox (bounding box) in pixels (e.g., x=100, y=200).

The detectionTo3D function performs un-projection:

- **Normalization**: It converts screen coordinates to a range of -1 to +1.
- **FOV Calculation**: It factors in the camera's Field of View.
- **Vector Ray**: It creates a direction vector from the user's head position toward the object.
- **Placement**: It places the 3D label on that vector at a defined distance (DIST).



7.3 Implementing Hit-Testing

Hit-testing allows the app to "fire" an invisible ray (raycast) into real space to detect intersections with floors or tables. When the ***hitTestSource*** returns a result, the cursor (green ring) is snapped to that pose (position and orientation).

7.4 AI Pipeline: Detection and Labeling

In the code, detection is not performed every frame (which would lag the headset), but every 2000ms (DETECT_INTERVAL_MS).

- **Label Sprite**: Since Three.js cannot render standard HTML fonts directly in a 3D scene, we use a CanvasTexture. We "draw" the text on an invisible 2D HTML canvas and apply it as a texture to a 3D Sprite that always faces the user (billboarding).

7.5 The Necessity of HTTPS

WebXR and camera access (getUserMedia) are classified as "Powerful Features" by browser vendors. They will only work in a **Secure** Context.

- **Localhost Exception**: You can test on your PC using http://localhost, but as soon as you try to access that server from your Quest 3 headset, the browser will block XR features because it sees an insecure network IP (e.g., http://192.168.1.10).
- **The Solution**: You must use a **Tunneling Service** or **Secure Hosting**.

7.6 Meta Quest Link & Developer Mode

To run and debug your code efficiently, your Quest 3 must be recognized as a developer device.

Step-by-Step Activation:

- **Developer Account**: Register at dashboard.oculus.com. You will need to create an "Organization" (it can be any name).
- Mobile App: Open the Meta Quest app on your phone, go to **Menu > Devices > Headset Settings > Developer Mode**, and toggle it **ON**.
- **The Link**: Connect your Quest 3 to your PC using a high-quality USB-C 3.0 cable or via Air Link (High-speed Wi-Fi 6).

7.7 The Three.js Foundation (The Boilerplate)

Before entering AR, we must initialize a standard 3D environment. However, for Quest 3, two settings are non-negotiable:

- **Alpha & Antialias**:

```
javascript
const renderer = new THREE.WebGLRenderer({ antialias: true, alpha: true });
```

- **XR Activation**:

```
enderer.xr.enabled = true;
```

This tells Three.js to listen for the Quest's head-tracking data (6DOF) and apply it to the `camera` object automatically.

7.8 Managing the AR Session (The Lifecycle)

The "Enter AR" button triggers **navigator.xr.requestSession**. This is where we define what "superpowers" our app needs from the Quest 3 hardware.

Required & Optional Features:

- **local-floor**: This tells the Quest to set the $Y=0$ coordinate at the actual physical floor level.
- **hit-test**: Enables the ability to raycast against real-world geometry.
- **plane-detection**: Requests the "Semantic" data (knowing which mesh is a 'table' vs. a 'wall').

```
JavaScript
const session = await navigator.xr.requestSession('immersive-ar', {
    requiredFeatures: ['local-floor'],
    optionalFeatures: ['hit-test', 'plane-detection']
});
```

8. AR + AI OBJECT DETECTION DEMO

8.1 Project Overview

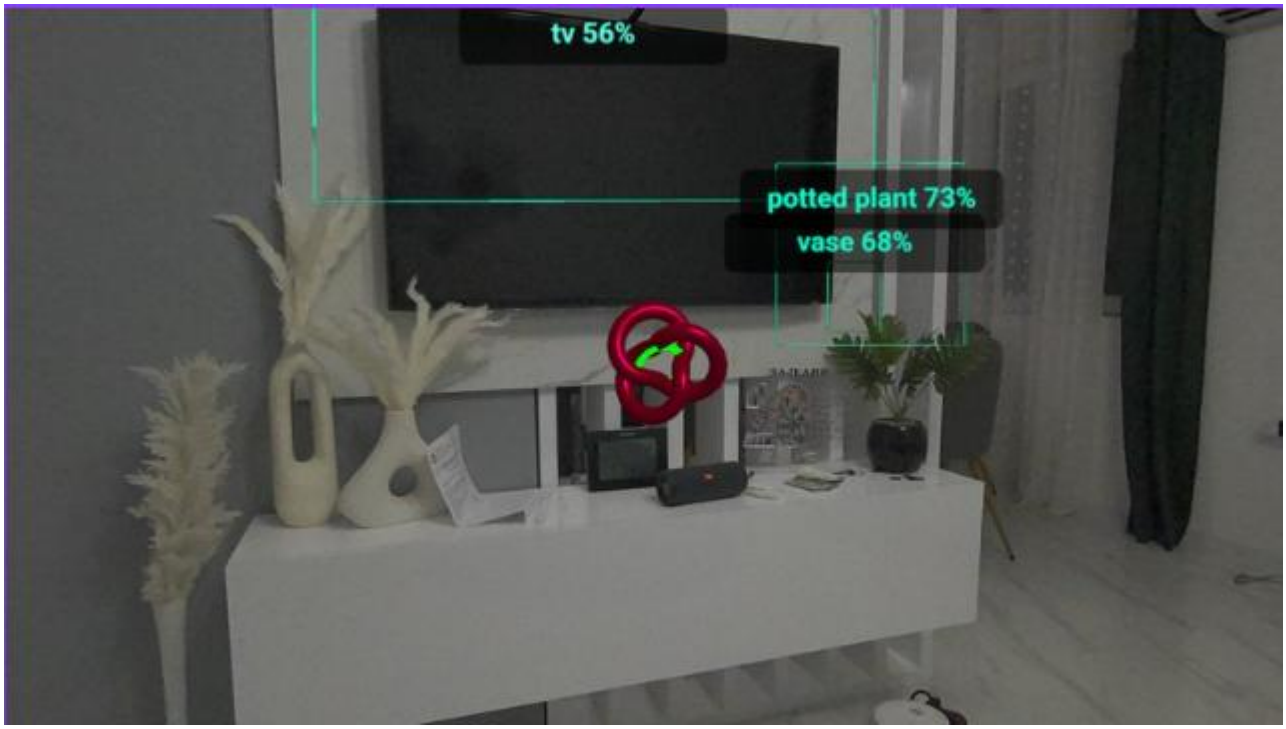
This is a WebXR Augmented Reality application for Meta Quest 3 that combines:

- **Passthrough AR** — the real room is visible through the headset cameras
- **Hit-test interaction** — a virtual cursor that snaps to real-world surfaces
- **AI object detection** — TensorFlow.js recognizes objects in the camera feed and displays labels in 3D space
- **Scene Understanding** — WebXR Plane Detection visualizes detected surfaces (floor, walls, table)

Live URL: http://92.113.18.92/

Single file:

index.html



8.2 Project Architecture

index.html (all-in-one)

├── HTML → canvas + UI button + hidden video element

├── CSS → transparent background, overlay UI

└── JavaScript

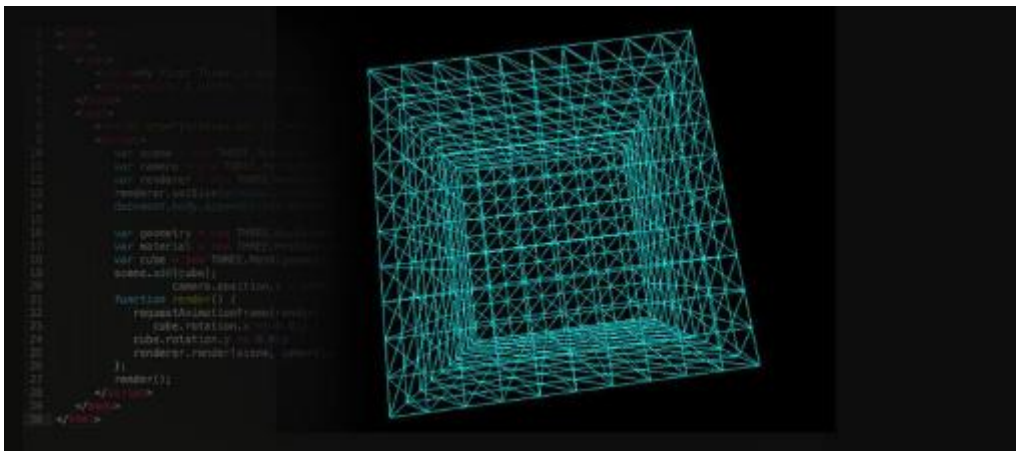
    ├── Three.js → 3D rendering engine (scene, camera, materials)

    ├── WebXR API → AR session, hit-test, plane detection

    └── TF.js → AI inference (coco-ssd model)

Why pure Three.js (no A-Frame)?

During development it was discovered that A-Frame has its own internal render loop that conflicts with an explicit **immersive-ar** WebXR session. A-Frame starts an **immersive-vr** session (opaque, no passthrough) instead of **immersive-ar**. Switching to pure **Three.js** gave us direct control over both the session type and the render loop.



8.3 Technology Stack

Technology	Version	Role
Three.js	0.157.0	3D rendering, geometry, materials
WebXR Device API	Browser-native	AR session, hit-test, plane detection
TensorFlow.js	4.15.0	In-browser ML inference engine
COCO-SSD	2.2.3	Pre-trained object detection model
getUserMedia API	Browser-native	Camera stream for TF.js analysis

8.4 Application Flow

Page loads

→ TF.js model (coco-ssd) loads asynchronously (~6MB)

→ Check: navigator.xr.isSessionSupported('immersive-ar')

→ "Enter AR" button becomes active

User clicks "Enter AR"

→ Request: navigator.xr.requestSession('immersive-ar', {...})

→ Quest enables Passthrough (camera visible through headset)

→ Simultaneously: getUserMedia() → camera stream for TF.js

→ renderer.setAnimationLoop() starts (XR render loop)

Every frame (~72fps on Quest 3)

- scene.background = null (ensures transparency)
- Hit-test → updates cursor position
- Plane Detection → updates surface visualization
- Every 2s → runDetection() → AI inference
- renderer.render(scene, camera)

8.5 Detailed Code Explanation

1. HTML Structure (lines 1–78)

```
HTML
<video id="tfvideo" playsinline muted autoplay></video>
```

A hidden <**video**> element receiving the ***getUserMedia***() stream. TensorFlow.js uses it as input for inference — it is never displayed to the user, only analyzed.

```
html
<div id="ui"> ... <button id="ar-button"> </div>
```

An overlay UI layer floating above the **Three.js** canvas. ***pointer-events: none*** on the container but ***pointer-events: all*** on the button only — so the overlay doesn't block 3D interactions.

2. Three.js Initialization (lines 84–109)

```javascript
const renderer = new THREE.WebGLRenderer({ antialias: true, alpha: true });
renderer.xr.enabled = true;
```

alpha: true creates a WebGL canvas with a transparent alpha channel — this is a prerequisite for passthrough. ***xr.enabled = true*** activates **Three.js**'s built-in WebXR integration.

```javascript
const camera = new THREE.PerspectiveCamera(70, aspect, 0.01, 20);
scene.add(camera);
```

The camera is **added to the scene**. This is important because objects attached to the camera (child entities) need to be in the scene hierarchy.

```javascript
const cursorGeo = new THREE.RingGeometry(0.04, 0.06, 32);
cursorGeo.rotateX(-Math.PI / 2);
```

The ring geometry is rotated -90° on the X axis so it lies flat horizontally on detected surfaces. Without this rotation it would stand vertically.

3. makeLabel() — Text Sprite (lines 113–138)

```javascript
function makeLabel(text, color, bgColor) {
  const canvas = document.createElement('canvas'); // 512x128 px
  // Draws rounded rect + text
  const texture = new THREE.CanvasTexture(canvas);
  const sprite  = new THREE.Sprite(material);
  sprite.scale.set(0.6, 0.15, 1); // 0.6m wide in 3D space
  return sprite;
}
```

THREE.Sprite is an object that always faces the camera (**billboarding**) — ideal for labels in 3D space. It is drawn on an HTML canvas, converted to a ***CanvasTexture***, and applied to a ***SpriteMaterial***.

depthTest: false ensures the label is always visible even if it is geometrically "behind" another 3D object.

4. detectionTo3D() — 2D → 3D Projection (lines 140–163)

This is the mathematical core of the AI-AR integration.

```javascript
// Normalize bbox center to [-0.5, 0.5]
const nx = (bbox.x + bbox.width/2)  / videoWidth  - 0.5;
const ny = (bbox.y + bbox.height/2) / videoHeight - 0.5;
// Apply camera FOV (70° horizontal)
const fovH = 70 * Math.PI / 180;
const dir  = new THREE.Vector3(
  Math.tan(nx * fovH),
  Math.tan(-ny * fovV), // flip Y (image top-down, WebGL bottom-up)
  -1                    // forward in camera space (Three.js uses -Z)
).normalize();
// Rotate into world space using the current XR camera orientation
dir.applyQuaternion(camera.quaternion);
// World position = camera position + direction × distance
return camera.position.clone().addScaledVector(dir, placeDist);
```

*Example: If the AI detects a chair in the left third of the video, **nx ≈ -0.17**. This converts to a negative angle (left of the gaze axis). The label is placed 1.8m in front of the camera in that direction.*

5. makeBox3D() and bbox2dSize3D() — 3D Bounding Boxes (lines 165–182)

```javascript
function makeBox3D(w, h, colorHex) {
  const edges = new THREE.EdgesGeometry(new THREE.BoxGeometry(w, h, 0.01));
  return new THREE.LineSegments(edges, material);
}
```

EdgesGeometry + LineSegments *renders only the edges of a box — the effect of a thin wireframe border with no filled surface.*

```javascript
function bbox2dSize3D(bboxW, bboxH, videoW, videoH, dist) {
  const fovH   = 70 * Math.PI / 180;
  const totalW = 2 * dist * Math.tan(fovH / 2); // total visible width at given dist
  return {
    w: (bboxW / videoW) * totalW,  // bbox fraction → meters
    h: (bboxH / videoH) * totalH
  };
}
```

*The **totalW** formula is standard perspective projection: **at distance d**, the visible width depends on the FOV. From this we derive how many meters correspond to the pixels of the bounding box.*

6. runDetection() — AI Inference Pipeline (lines 205–247)

```javascript
async function runDetection() {
  const predictions = await cocoModel.detect(tfVideo);
  // Clear old labels and boxes
  activeLabels.forEach(({ sprite, box }) => { scene.remove(sprite); scene.remove(box); });
  activeLabels = [];
  predictions.forEach(pred => {
    if (pred.score < 0.5) return; // Confidence threshold: 50%
    // ...create sprite + box...
    activeLabels.push({ sprite, box, expireAt: Date.now() + 4000 });
  });
}
```

***cocoModel.detect(tfVideo)** accepts a <video> element directly — TF.js internally reads pixel data from the current video frame.*

***pred.bbox** format: **[x, y, width, height]** in pixels.*

*Each detection creates a **JS(sprite, box)** pair that lives for 4 seconds.*

7. WebXR Session — Key Details (lines 274–285)

```javascript
const session = await navigator.xr.requestSession('immersive-ar', {
  requiredFeatures: ['local-floor'],
  optionalFeatures: ['hit-test', 'plane-detection']
});
renderer.xr.setReferenceSpaceType('local-floor');
await renderer.xr.setSession(session);
```

Why **immersive-ar** and not **immersive-vr**?

- **immersive-vr** = opaque black background (VR helmet experience)
- **immersive-ar** = passthrough camera + virtual content overlaid on top

local-floor reference space fixes the coordinate system to the room floor — **y=0** is at floor level.

hit-test and **plane-detection** are optional because they are not supported on all devices and browser versions — declaring them as optional prevents the session from failing if they're unavailable

8. Render Loop (lines 335–395)

```javascript
renderer.setAnimationLoop(function(time, frame) {
  scene.background = null;              // Ensures transparency every frame
  renderer.setClearColor(0x000000, 0); // Alpha = 0 (fully transparent)
  // ...hit-test, plane detection, AI...
  renderer.render(scene, camera);
});
```

Why **setAnimationLoop** and not **requestAnimationFrame**?

The **Three.js** XR render loop must be integrated with the WebXR frame callback. **renderer.setAnimationLoop()** automatically binds to the XR session when **renderer.xr.enabled = true**. The alternative (manually calling **session.requestAnimationFrame()**) requires calling **renderer.render()** manually but risks missing the correct XR view matrices that **Three.js** applies internally.

scene.background = null must be called **every frame** because Three.js may reset this value internally during certain operations.

9. Plane Detection (lines 355–378)

```javascript
session.detectedPlanes.forEach(plane => {
  if (!detectedPlanes.has(plane)) {
    // New surface detected — create a mesh
    const label = plane.semanticLabel; // 'floor', 'wall', 'table'...
    const mesh = new THREE.Mesh(PlaneGeometry, transparentMaterial);
    scene.add(mesh);
    detectedPlanes.set(plane, mesh); // store reference
  }
  // Every frame, update the surface pose
  const pose = frame.getPose(plane.planeSpace, xrRefSpace);
  mesh.position.copy(pose.transform.position);
  mesh.quaternion.copy(pose.transform.orientation);
});
```

plane.planeSpace is an XRSpace that tracks the physical surface. **frame.getPose()** returns its pose in the chosen reference space.

The **detectedPlanes** map **(Map<XRPlane, THREE.Mesh>)** prevents creating duplicate meshes for the same surface across frames.

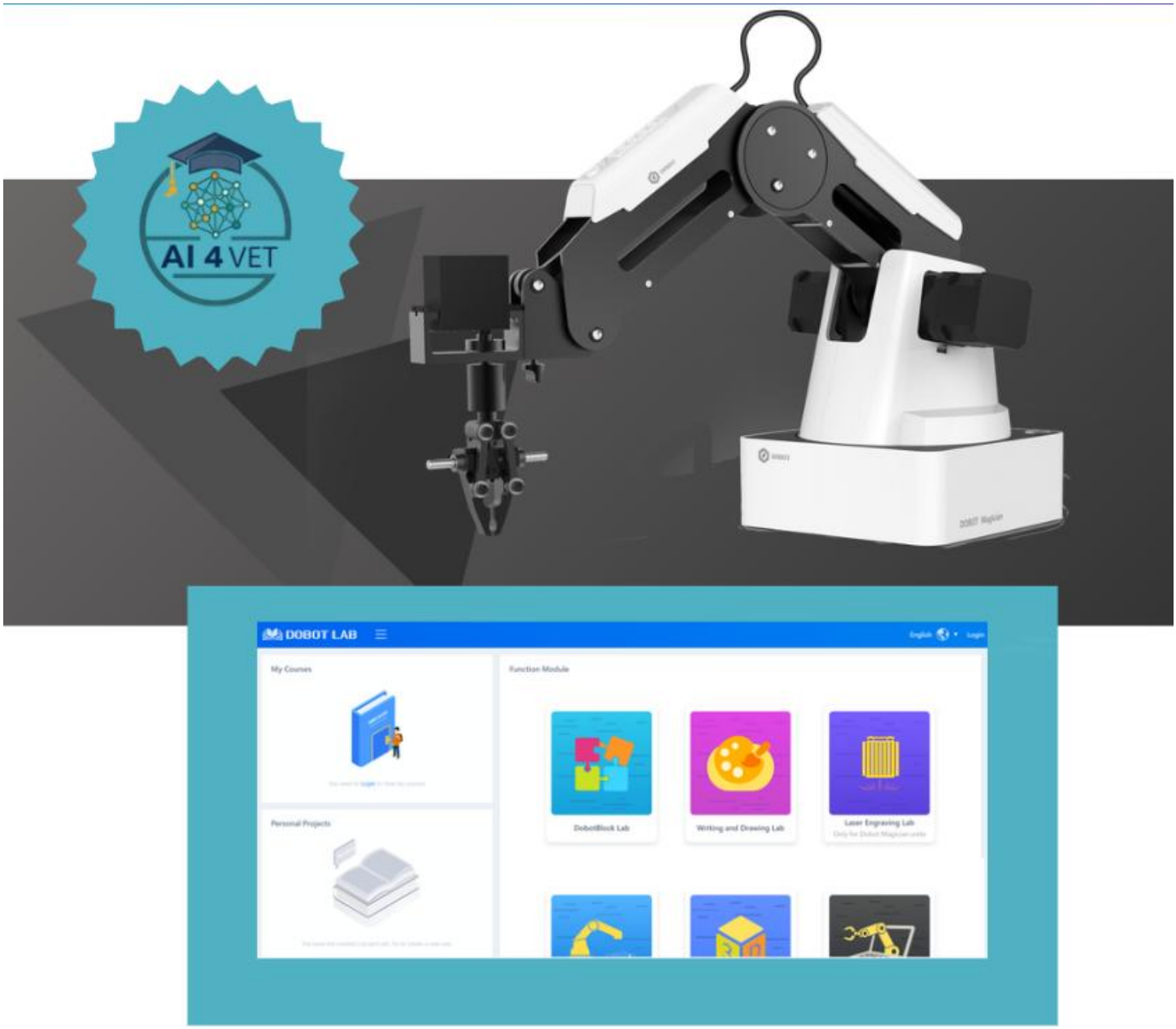
8.6 Known Limitations

Limitation	Reason
AI labels are not pixel-perfect aligned with object	The Quest passthrough cameras and getUserMedia deliver different streams with different FOV and optical calibration
Plane detection requires Quest Room Setup to be completed	The headset must have previously scanned the room
getUserMedia may be denied on some devices	Depends on browser permissions
AI inference is not real-time (runs every 2s)	coco-ssd is a relatively heavy model; lighter alternatives (MobileNet SSD) would give higher throughput

8.7 Source code

https://github.com/bciric1/ARVR

http://vr.aiforvet.eu/



III. AI WITH Dobot

9. DRIVER INSTALLATION INSTRUCTION

When first connecting the Dobot, connect the Dobot to PC through the USB port. Then, power on the Dobot, then the system will recognize the corressponding hardware automatically and will search for the correct driver and install it. However, if it fails to install, you can install it again manually. The installation flow chart as follows:

9.1 Download CH340 driver package and install it

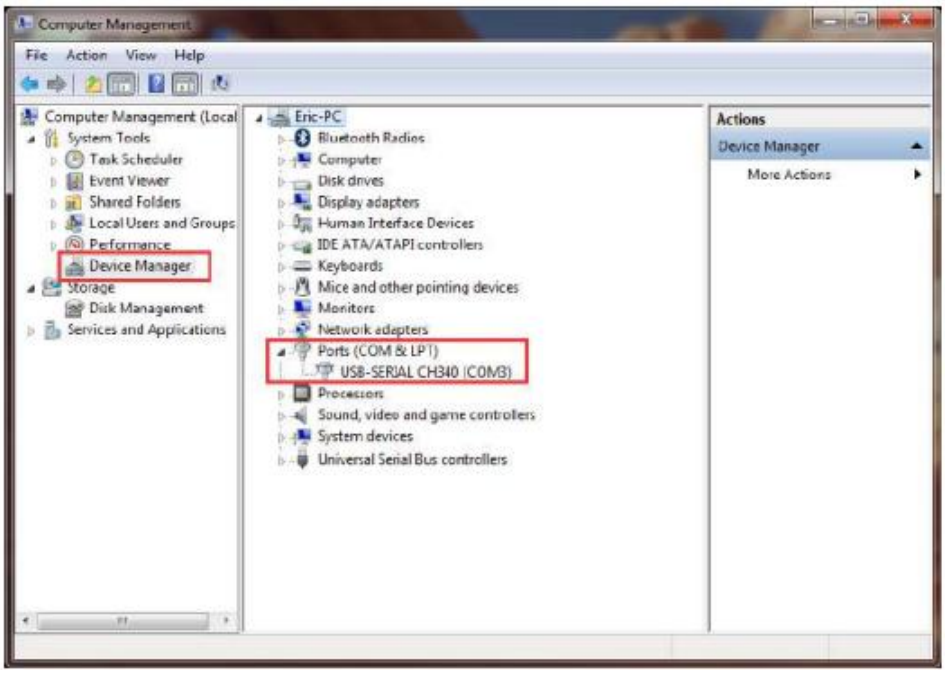
There are two versions of the driver based on Windows/Linux, so please choose the corressponding version to your OS to download. The driver can be located at the download address：

http://www.dobot.cc/downloadcenter.html?sub_cat=70#sub-download

After downloading, unzip and install the driver

9.2 Check if the equipment can work properly in the device manager

Open the device manager and if you can find the corresponding COM port of "USBSERIAL CH340", then it shows the driver is installed successfully. The correct installation is shown below:



10. DOBOTSTUDIO OPERATING INSTRUCTIONS

The software used by Dobot Magician is DobotStudio, and you can download the latest version from official website:
http://www.dobot.cc/downloadcenter.html?sub_cat=70#sub-download
After the file successfully downloads, unzip and double click DobotStudio.exe.

Select the corresponding Dobot serial port, by the top left corner of DobotStudio, and click "Connect". After a successful connection, "Disconnect" will be shown. When the Dobot is connected, the coordinate parameters will be updated on the right side of the interface

There are eight modules on the main software interface:

- Teaching & Playback: A system to teach the Dobot how to move. It enables the Dobot to accomplish recorded movements by manual control.
- Write & Draw: Controls the Dobot to write, draw or laser engrave.
- DobotBlockly：Teaches basic programming through a puzzle interface. Intuitive and easy to understand.
- Script: Edit scripting language to control the Dobot.
- LeapMotion: Control the Dobot by gesture.
- Mouse: Control the Dobot by mouse.
- LaserEngraving: Engraves images, shapes, and words through bitmap with the Dobot.
- Add More: Add even more functions for the Dobot!

10.1 Linear mode

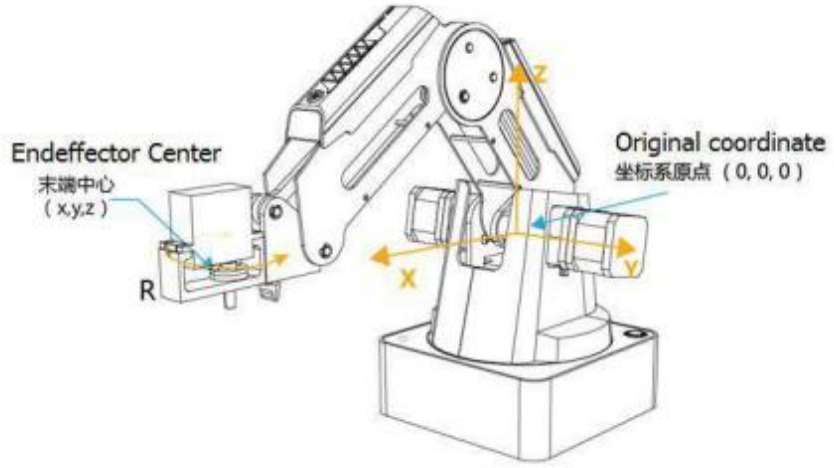
Based on the body axes coordinate system X, Y, Z with the origin at the center of three motors. X, Y, Z is the coordinate of the center of the end platform, and the direction of X is perpendicular to the base forward, Y is perpendicular to the base towards the left, and Z is vertical upward. R indicates the rotation of the servo joint relative to the coordinate frame (counter-clockwise is the positive direction).

（1） Click X+,X- and Dobot will move along X in the negative or positive direction;

（2） Click Y+,Y- and Dobot will move along Y in the negative or positive direction;

（3） Click Z+,Z- and Dobot will move along Z in the negative or positive direction;

（4） Click R+,R- and Dobot will move along R in the negative or positive direction.



10.2 Jog mode

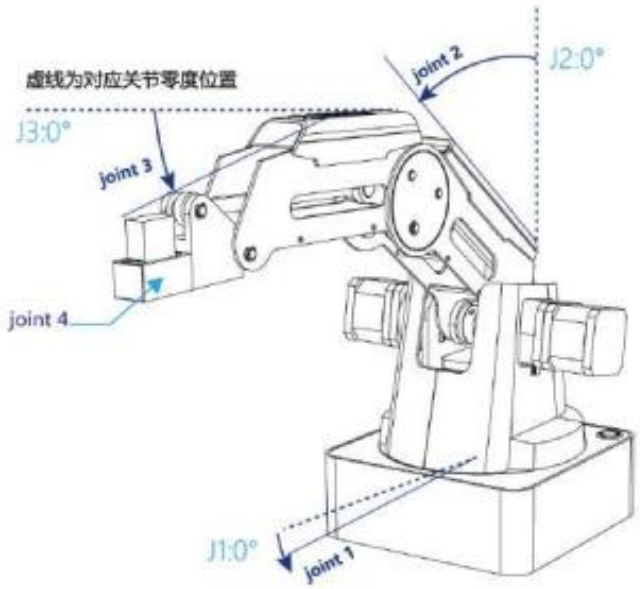
This movement is aimed for a single axis. Hold down the button, and the corresponding axis will move independently. Once the axis is maxed out, the joint will stop moving. Each axis has counterclockwise as the positive direction. Joint1、2、3、4 refer to base, rear arm, forearm and servo respectively.

（1） Click Joint1+、Joint1- and control the Dobot base motor to rotate in the negative or positive direction;

（2） Click Joint2+、Joint2- and control rear arm motor to rotate in the negative or positive direction;

（3） Click Joint3+、Joint3- and control forearm motor to rotate in the negative or positive direction;

（4） Click Joint4+、Joint4- and control servor to rotate in the negative or positive direction; Among this, the rotation range of Joint4 is ±150°

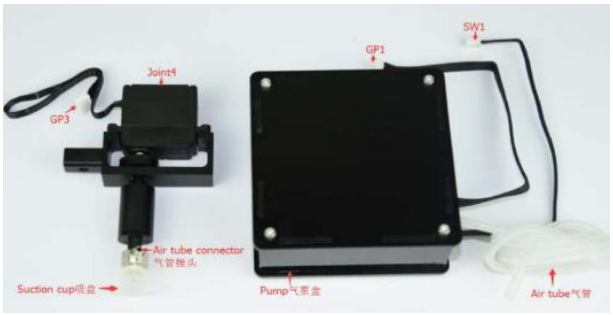


11. ACTUATORS ON DOBOT

Here we will learn how to suck up or grab simple objects using the function from menu Teaching & Playback. Because we need to use the air pump kit for the suction cup and the gripper kit, we will introduce these two kits together.

11.1 Air pump kit

The default installation of the Dobot Magician is the suction cup. The pump box and suction cup kit is shown below:



11.2 Pneumatic Gripper Kit

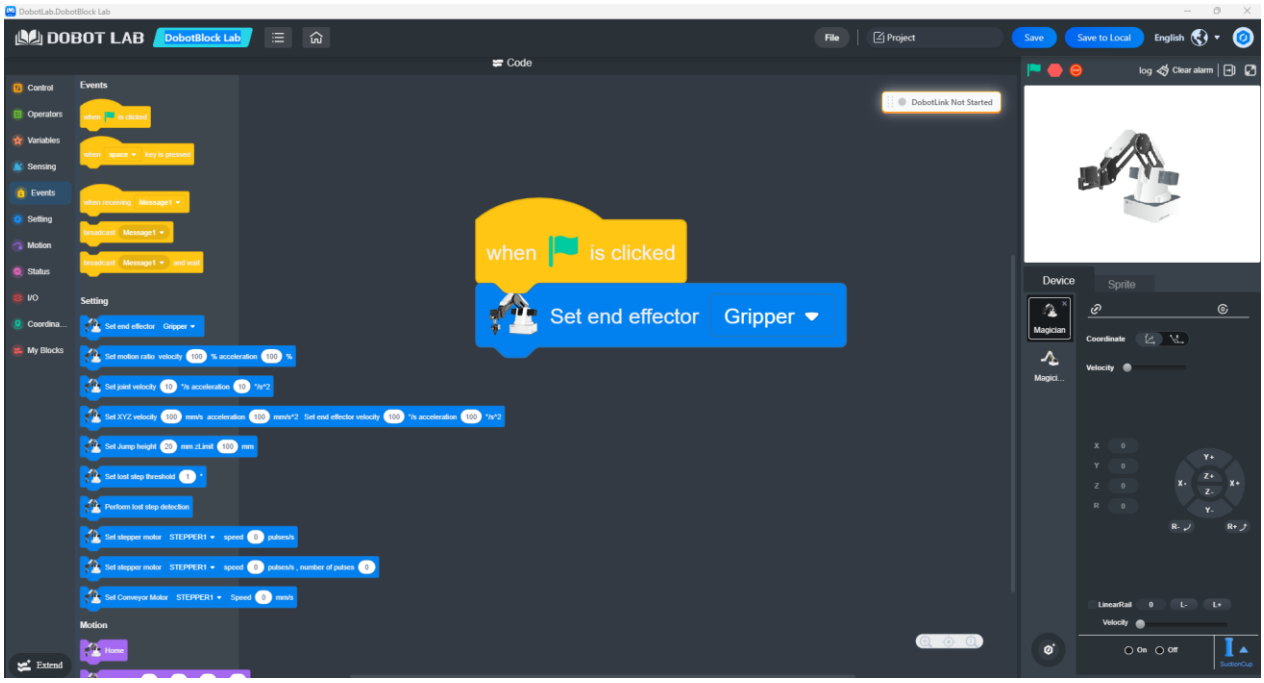
Pneumatic accessories are shown in following picture:

12. PROJECT: AUTOMATED WASTE CLASSIFICATION AND SORTING SYSTEM USING DOBOT MAGICIAN

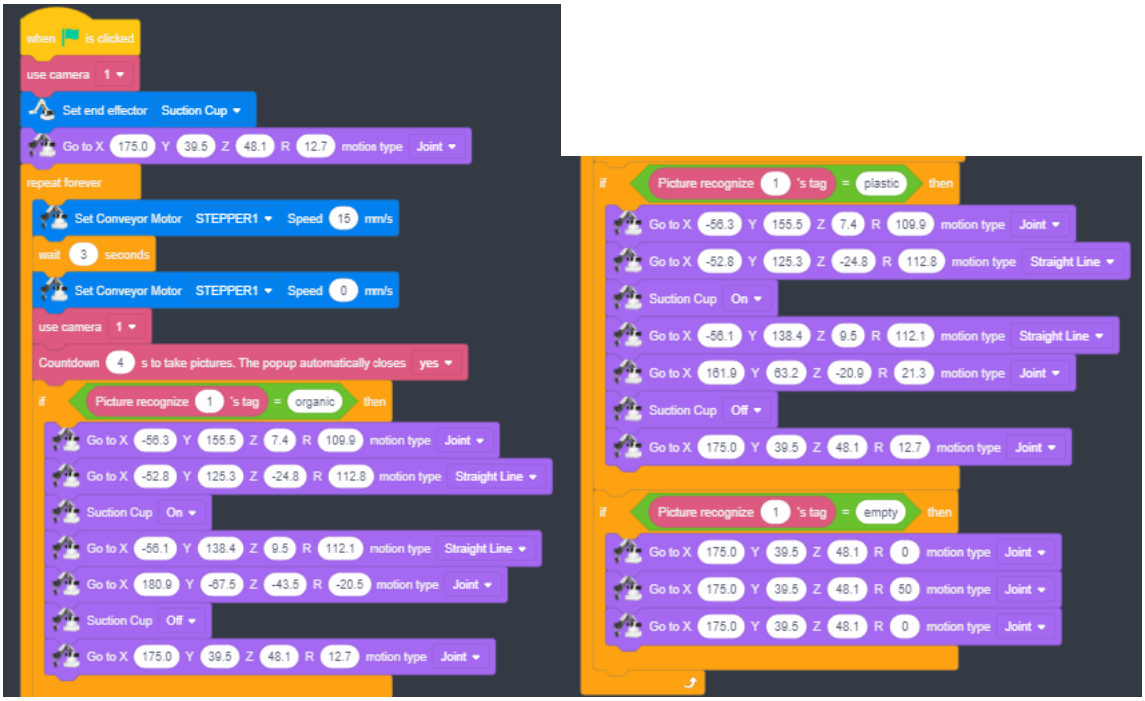
Dobot Blockly is a programming platform based on Google Blockly. In this process, users can program through the puzzle format, which is straightforward and easy to understand. Also, users can use the integrated API of Dobot anytime.

12.1 Blockly Interface

Open DobotStudio and click DobotBlock Lab:



12.2 Block code



12.3 Problem Initialization and Definition

Context and Motivation

In modern industry and ecology, manual waste sorting is a slow, inefficient, and often hazardous process. The goal of this project is to create an autonomous system that utilizes **Computer Vision** and **robotic manipulation** to recognize and physically separate different materials (in this case, organic waste and plastic).

Problem Statement

The primary challenge lies in the synchronization of three independent subsystems:

1. **Transport System:** A conveyor belt that brings objects into the workspace.
2. **Sensing System:** A camera that must identify the item type in real-time.
3. **Actuation System:** A robotic arm that must precisely execute picking and placing based on sensor feedback.

12.4 Solution Architecture

The solution is based on the **Dobot Magician** platform and a block-based programming environment (Blockly) that integrates AI modules for image recognition.

Hardware Components

- **Dobot Magician Arm:** A high-precision 4-axis robot.
- **Suction Cup:** Used as the end effector for its versatility in handling various shapes.
- **USB Camera:** Mounted above the belt for a stable top-down view.
- **Conveyor Belt:** Controlled via a stepper motor connected to the Dobot.

Software Logic

The algorithm follows an **iterative closed-loop model**: Start Belt -> Stop -> Capture -> Analyze -> Sort -> Return to Home.

12.5 Detailed Code Block Analysis

The program is divided into four key segments:

Configuration (Setup)

- **use camera 1**: Declares the hardware input for visual data.
- **Set end effector [Suction Cup]**: Soft-maps the air pump control to the robot's output.

- **Go to X:175, Y:39.5, Z:48.1**: This is the "Safety Position." The robot retreats to avoid obstructing the camera and ensures an optimal path to any point on the belt.

Logistics (Conveyor Control)

Inside the repeat forever block:

- **Set Conveyor Motor [STEPPER1] Speed 15 mm/s**: Activates the belt at a controlled speed to prevent light objects from sliding.
- **wait 3 seconds**: A critical time constant defining the distance between items.
- **Speed 0**: Stops the belt so the robot can perform a precise "static pick".

AI Inspection (Picture Recognition)

- **Countdown 4 s**: Provides stabilization time for the camera to focus and for the AI popup to process the image.
- **If Picture recognize 1's tag = [organic/plastic]**: Calls the Deep Learning API to compare the current frame against a trained dataset and returns a class tag.

Kinematics and Manipulation

For each branch (organic/plastic), the system uses two motion types:

1. **Joint Motion (PTP):** Moves all joints simultaneously for the fastest point-to-point travel.
2. **Straight Line (Linear):** Moves the arm strictly vertically on the Z-axis to ensure the suction cup makes perfect contact without knocking the item over.

12.6 Operational Algorithm (Step-by-Step)

- **START:** Initialize instruments and set end effector.
- **TRANSPORT:** Move belt for 3 seconds, then stop.
- **SENSE:** Capture image and identify tag via AI.
- **DECISION:**
- **If Organic:** Move to grab coordinates, turn suction **On**, move to organic bin, turn suction **Off**.
- **If Plastic:** Move to grab coordinates, turn suction **On**, move to plastic bin, turn suction **Off**.
- **If Empty:** Perform a signaling "shake" motion (rotating the R-axis).
- **RESET:** Return to the initial safety coordinates.
- **LOOP:** Repeat the cycle indefinitely.

12.7 Technical Coordinate Specifications

Based on the script, the following coordinates have been calibrated:

Point	X (mm)	Y (mm)	Z (mm)	R (°)	Description
Home	175.0	39.5	48.1	12.7	Idle/Wait position
Approach	-56.3	155.5	7.4	109.9	Position above item
Grab	-52.8	125.3	-24.8	112.8	Contact point (Lowered)
Organic Bin	180.9	-67.5	-43.5	-20.5	Disposal for organics
Plastic Bin	161.9	63.2	-20.9	21.3	Disposal for plastics

12.8 Implementation Troubleshooting

- **Inaccurate Grabbing:** Ensure the item is centered. If it varies, use guides on the belt or integrate dynamic X/Y feedback from the AI camera.
- **Classification Errors:** AI recognition is sensitive to light. Use a high-contrast belt color and consistent external LED lighting.
- **Safety:** Always ensure the workspace is clear of obstacles before starting the repeat forever loop.

IV. AI WITH rPI 5

13. INTRODUCTION

The Raspberry Pi is a little computer about the size of a deck of cards and capable of running a full Linux desktop operating system while consuming only modest power. It includes USB ports for connecting a keyboard and mouse along with a variety of other peripherals, an ethernet adapter, and HDMI monitor connections. The Raspberry Pi was originally constructed for education, but has found fruitful uses for hobbyists, home automation, industrial applications, and as an appropriate technol- ogy for use in schools in the majority world. It is manufactured to comply with RoHS (Restriction of Hazardous Substances) directives and relies on a single microSD card for its storage. It runs a variant of Linux called the Raspberry Pi OS, and supports a wide variety of open source software, including a plethora of educational programs. Moreover, it can be purchased at a modest price. This guide was written primarily with engineering and computer science students in mind, but it will be of interest to others who have a keen interest in learning more about programming and technical computing.

13.1 Initial Setup of the Raspberry Pi

Insert an SD card with the Raspberry Pi OS in the Raspberry Pi and apply power. When you first boot the regular Raspberry Pi OS it will be running a graphical desktop environment with a friendly menu- driven interface. Upon the first boot, a dialogue box will appear guiding you through an initial setup. Follow the prompts to configure the keyboard and username. Provide a username and a password as prompted. Configure your Wi-Fi settings and select the option to "Update Software" (note that this may take a very long time when you first setup the Raspberry Pi).

13.2 Getting Started with the Command Line

There is also a command-line based version of the operating system called Raspberry Pi OS Lite. This version of the OS consumes less power than the regular Raspberry Pi OS running a desktop en- vironment and can be used on older models of the Raspberry Pi with less RAM. When setting up the Raspberry Pi with the Lite OS, you will be prompted to configure the keyboard and provide a user- name and password. The Lite version of the OS is well suited for using the Raspberry Pi as a server, as an embedded system, or in an IoT (Internet of Things) application. However, for regular desktop use, the regular version of Raspberry Pi OS is best. When using the Desktop OS, the command line can still be accessed using the Terminal app. Alternately, it can also be accessed using a virtual console by pressing CTRL+ALT+F1 which will enter a full screen terminal. If a login prompt appears, you can login using the username and password you configured during setup. One can return to the desktop from a virtual console by pressing CTRL+ALT+F7. Additional virtual consoles can be independently accessed using CTRL+ALT along with the keys F2 through F6.

13.3 The Shell

Once you enter the command line, you will be running in a Linux shell. In simple terms, a shell is a command interpreter which provides a rich set of commands which can be used to execute programs and interface with the operating system. The Raspberry Pi OS uses the Bash (Bourne Again Shell) by default, a shell based on an older shell called the Bourne Shell. BASH is popular among users of Linux and features automatic command line completion using the tab key and can be used to create programs called shell scripts. There are a variety of different Linux shells that can be used. As indicated, the default Linux shell is the Bash shell, but other shells are also available. Each shell has its own features and options. For example, to switch the default shell from Bash to the Z shell (zsh), type the following

```
sudo apt install zsh -y
chsh -s /bin/zsh
```

After issuing these command, logout and then back in and you should now be running with zsh. Various configuration options can be set inside a file named .zshrc located within your home folder.

13.4 Shell commands

Some of the commands available in the shell are summarized below:

Command	
cd directory	Changes the current working directory to directory
pwd	Displays the name of the current working directory

mkdir directory	Create a new directory called directory
rmdir directory	Removes the directory called directory
ls	Display a list of files in the current directory
cp f1 f2	Copy a file from the source f1 to the destination f2
rm filename	Remove a file filename
mv f1 f2	Move a file from f1 to f2
ftp host	Transfer files to and from host

These commands only represent a portion of the user commands available in a Linux shell. An on-line manual referred to as the man (manual) pages provides help on the many commands and programs that can be called from the shell. The syntax for invoking the man utility is as follows:

```
man command-name
```

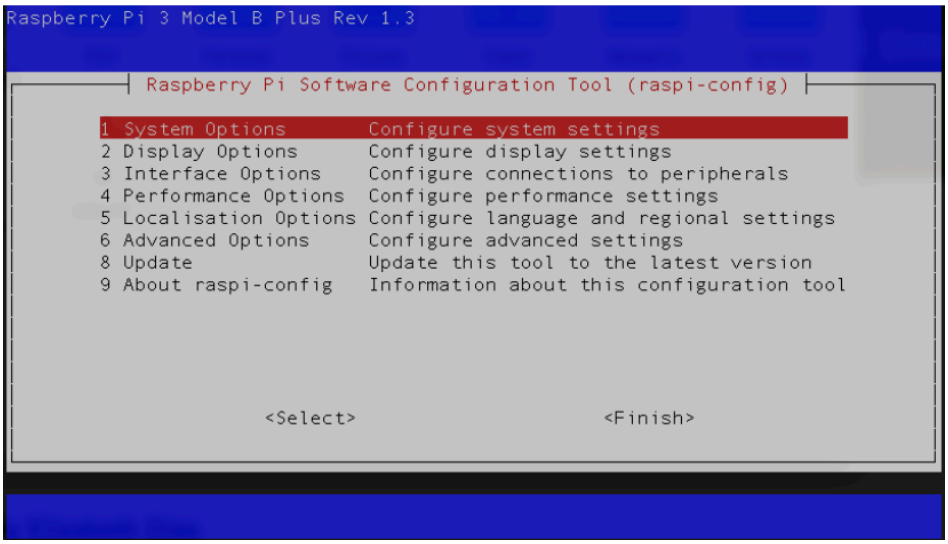
The information regarding the specified command-name will then be displayed on the screen.

13.5 Configuring the Raspberry Pi OS

The Raspberry Pi OS comes with a utility called raspi-config that can be used to configure a wide variety of settings and services. To run this utility, type:

```
sudo raspi-config
```

This will launch the Raspberry Pi configuration utility within the terminal with a main menu like the one shown below:



The Raspberry Pi configuration utility can be used to enable the camera interface as well as serial, I2C, and SPI communications. It also includes an option to boot directly to the command line rather than the desktop.

13.6 Connecting Remotely to the Raspberry Pi

It is possible to run the Raspberry Pi headless, without a screen, keyboard, or mouse. This is often the case when using the Raspberry Pi in an embedded application or an IoT (Internet of Things) configuration. The following subsections describe how to connect to your Raspberry Pi remotely using one of the following options:

- using a USB-to-TTL serial cable
- using SSH over an Ethernet or Wi-Fi connection
- Raspberry Pi Connect service

Some models of the Raspberry Pi can be connected using a USB cable. This works by enabling USB Gadget Mode, which allows a USB port to present itself as a variety of different types of devices. This is known to work with the Raspberry Pi Zero and not with most other models. The two approaches described below should work with all models of the Raspberry Pi.

14. INTRODUCTION TO PROGRAMMING LANGUAGES

The Raspberry Pi repositories include support for COBOL, as well as a rich set of more modern programming languages such as Python, C, C++, and Java. It has support more niche and legacy pro- gramming languages. Generally, the paradigms for programming languages fall into three general categories:

• procedural programming languages

• object oriented programming languages

• functional programming languages

14.1 The Python Programming Language

Python was invented in the early 1990's by Guido van Rossum. Python can be written using either a procedural or object oriented paradigm. It is an open source project that is widely available, uses simple syntax and includes rich libraries and offers a variety of programming tools, Python source code is run on a virtual machine which translates code into the specific machine-code executed by the processor. Because Python is interpreted by a virtual machine, it is platform independent.

The Raspberry Pi should have Python installed by default and can run Python programs directly from the command line. To enter a program, you first need a plain text editor. If you are using a desktop environment, there is a friendly, graphical, integrated development environment (IDE) for Python suitable for beginners called Thonny. To install Thonny, type:

```
sudo apt install thonny
```

For more advanced users in a graphical environment, the vscode program provides an excellent editor for coding in a variety of different languages, including Python. As described earlier, vscode can also be run to edit files remotely. If you are using the command line, you can use any of the command line editors described in the preceding sections, including vi, emacs, or nano. For example, to edit a Python source file called hello.py, type the following:

```
nano hello.py
```

Next, enter the following code into the source file:

```
name = input('What is your name? ')
print('Hi', name,' welcome to the Raspberry Pi!')
print('Good Bye')
```

Next, save and exit nano and run the file by typing:

```
python3 hello.py
```

The program should run as expected.

14.2 Compiling and Running a C/C++ Program

Linux has a variety of tools to support software development in both C and C++. In fact, the Linux operating system itself is written in C. The C programming language is a procedural language, and C++ builds on C to provide support for object oriented programming. The compilers which we will use under Linux are the GNU C Compiler (gcc) and the GNU C++ compiler (g++). To ensure the GNU C/C++ compiler tools are installed, type:

```
sudo apt install gcc g++ gdb build-essential
```

For example, to enter a simple C program named hello.c using the nano editor, type the following:

```
nano hello.c
```

Using the editor, enter the following code into the source file:

```
/* A Raspberry Pi C program */
#include <stdio.h>
int main(void)
{
    printf("Hello world.\n");
    printf("Compiled and run on a Raspberry Pi.\n");
    return 0;
}
```

Save and exit the editor. To compile your source code type the following at the prompt in a terminal window. For example, to compile the hello.c program above, you can enter:

```
gcc -Wall -o hello hello.c
```

The gcc compiler has numerous other command line options which you may use. For more information consult the man pages. Note that the C++ compiler can be invoked by using g++ in place of gcc. To run the compiled program in the current working directory, do not forget to specify a ./ in front of the program name to specify the path as the current directory. For example, to run the hello.c program after compiling it as ini, type:

```
.\hello
```

If no output filename was provided to the compiler the default output filename will be a.out.

14.3 Compiling and Running Java Program

It is also possible to develop and run Java programs using Linux on the Raspberry Pi. There are two different packages which can be installed: on provides the Java Runtime Environment (JRE) and another provides the Java Development Kit (JDK). The JRE just allows you to run Java programs, but the JDK enables one to compile and run Java programs. To install the OpenJDK Java development kit, type the following:

```
sudo apt install default-jdk
```

Once this is installed you can compile a Java program. For example, enter the following simple Java program named hello.java using a plain text editor:

```
/* A Java program */
class Main {
    public static void main(String args[]) {
        System.out.println("Hello world.\n");
    }
}
```

Compile the program by typing the following at the prompt:

```
javac hello.java
```

where myprogram.java is the name of a Java source file. To run a Java program, type the following at the prompt: **java Main** where Main is the name of the class where the program begins.

15. EXPLORING ARTIFICIAL INTELLIGENCE

15.1 Hardware and Software Support for AI

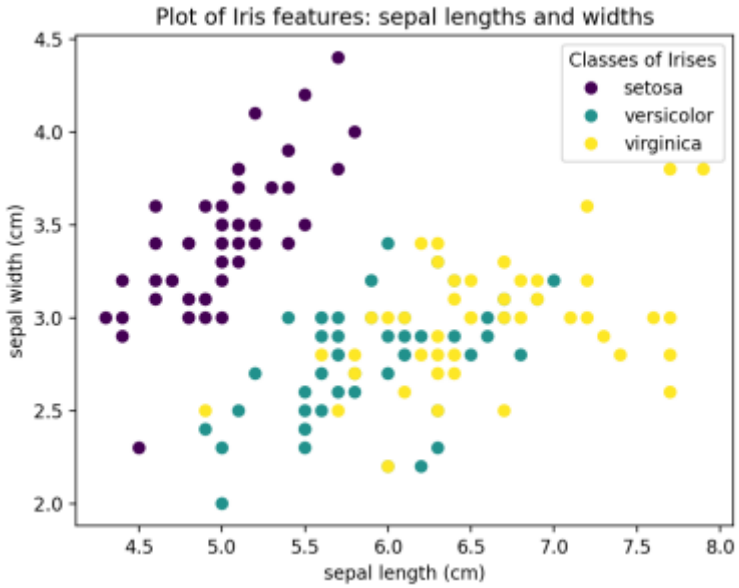
Recent models of the Raspberry Pi processor include multiple ARM cores. Despite its respectable hardware capabilities, the Raspberry Pi may struggle with the computational demands of more demanding artificial intelligence (AI) applications. For more demanding applications, there is a Raspberry Pi add-on board — referred to as a HAT (Hardware Attached on Top) that provides an NPU (Neural Processing Unit) to accelerate machine learning computations. The Raspberry Pi AI HAT+ provides a high-performance, power-efficient AI processor for the Raspberry Pi 5. Besides hardware support, there are a wide variety of powerful machine learning libraries which can be used with the Raspberry Pi, including scikit-learn and LiteRT. Python includes a number of powerful supporting libraries, such as Matplotlib and plotly for plotting,NumPy for providing fast operations on arrays, and Pandas for data analysis and manipulation. The following sections provide examples of some of these libraries in action.

15.2 SciKit Learn

SciKit Learn is an open source machine learning library that works with Python. SciKit Learn provides many different features, including regression and clustering algorithms, Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and Support Vector Machines (SVMs). All forms of machine learning require a dataset which is used for training. SciKit includes a selection of toy datasets that can be used to experiment with the machine learning libraries. The following sections demonstrate PCA and SVMs using the classic iris flower dataset which is part of the collection of toy datasets. This iris dataset was originally compiled by biologist Ronald Fisher in a wellknown 1936 paper. This dataset comprises samples of three species of the Iris flower (Iris setosa, Iris virginica, and Iris versicolor) and measurements of the length and the width of both the sepals1 and the petals. The SciKit Learn library includes this iris dataset for testing and demonstration purposes. The followig Python code uses Matplotlib to plot the sepal width versus the sepal length for each of the three iris classes in the dataset.

```
from sklearn import datasets
import matplotlib.pyplot as plt
# Load the classic iris dataset
iris = datasets.load_iris()
# Plot sepal lengths vs. widths for irises in dataset
fig, ax = plt.subplots()
scatter = ax.scatter(iris.data[:, 0], iris.data[:, 1], c=iris.target)
ax.set_title("Plot of Iris features: sepal lengths and widths")
ax.set(xlabel=iris.feature_names[0], ylabel=iris.feature_names[1])
fig = ax.legend(scatter.legend_elements()[0], iris.target_names, loc="best
", title="Classes of Irises")
plt.show()
```

Running this code results in the following plot of the iris dataset:



15.3 SVM Image Classification

The following example is inspired by the SciKit example on recognizing hand-written digits and uses the hand-written digits dataset from the UC Irvine machine learning repository. This dataset has 1797 image samples comprised of an $8x8$ array of pixels with 10 classes where each class refers to one digit (0-9). A short Python program to load the hand-written digits dataset and display them is shown below:

```
import matplotlib.pyplot as plt
from sklearn import datasets, metrics, svm
from sklearn.model_selection import train_test_split

# load hand-written digits dataset
digits = datasets.load_digits()

# display a sample of ten hand-written digits in the dataset
fig, axes = plt.subplots(nrows=2, ncols=5)
for ax, digit in zip(axes.flatten(), digits.images):
    ax.set_axis_off()
    ax.imshow(digit,cmap='gray')
fig.tight_layout()
plt.show()
```

The plot showing ten hand-written digits in the dataset is shown below.

We can split the hand-written digits dataset into two sets: a training set and a set for testing. SciKit has a function for taking the larger digits dataset and splitting it in half into training and test sets as follows:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
```

where X is an array of the features and y is a vector of labels that classify the data. Hence we can proceed to train an SVM using half the dataset for training and then use the remaining half of the dataset for testing. The accuracy of the SVM can be determined by comparing the digits predicted by the SVM with the actual labels for the hand-written digits and the recognition rate can be reported. The recognition rate is the total number of correctly identified digit images divided by the total number of test images. The following code defines an SVM classifier based on half of the dataset and then determines the recognition rate using the other half of the dataset as test images.

```
import matplotlib.pyplot as plt
from sklearn import datasets, metrics, svm
from sklearn.model_selection import train_test_split

# Read digits and reshape digits as a vector of images
digits = datasets.load_digits()
```

```
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

# Create a support vector machine classifier
svm = svm.SVC()
# Split the dataset: half for training and half for testing
X_train, X_test, y_train, y_test = train_test_split(
data, digits.target, test_size=0.5, shuffle=False)

# Train on hand-written digits in the training set
svm.fit(X_train, y_train)
# Predict the value of the digit using the testing set
predicted = svm.predict(X_test)

# compute the recognition rate
matches = 0
for x in range(len(y_test)):
if y_test[x] == predicted[x]:
matches += 1
recognition_rate = (matches/len(y_test)) * 100;
print(f'Recognition rate: {recognition_rate:.2f}%')
```

The output of this code shows the following:

Recognition rate: 96.11%

This indicates a respectable recognition rate using an SVM for handwritten digit classification. For more details, we can plot a confusion matrix to visualize the accuracy of a machine learning algorithm. The confusion matrix is organized into rows and columns: each row represents each of the classes in a dataset and each column represents the predictions that were made. Ideally, the actual classes and the predictions will perfectly align such that the diagonal of the confusion matrix is populated with 100's and with 0's everywhere else. The diagonal martix corresponds to true recognition rates whereas all other cells represent the occurrences of false classifications.

16. EXAMPLES

16.1 Build Your Own "Parent Detector" Security System

Project Overview: Have you ever wondered who sneaks into your room when you aren't around? In this project, you will build a smart security system—often called a "Parent Detector" to find out exactly who has been in your room.

You will achieve this by combining a Raspberry Pi, a motion sensor, and a camera to automatically trigger and record video footage of any intruders.

Step 1: Gather Your Hardware

To build this automated security camera, you will need three main hardware components:

- A Raspberry Pi: This acts as the brain of your project, processing the signals and running the code.
- A Passive Infrared (PIR) Motion Sensor: This component acts as the "eyes" of your system, detecting changes in infrared heat to sense when a person moves nearby.
- A Raspberry Pi Camera Module: This is the device that will capture the video evidence of the intruder.

Step 2: Understand and Tune the PIR Sensor

Before plugging anything in with wires, you need to configure the physical settings on the PIR sensor itself. If you look at the back of the PIR module, you will see two orange components with small cross-shaped sockets that perfectly fit a Philips screwdriver.

These orange dials are called potentiometers, and they allow you to manually adjust how the sensor behaves:

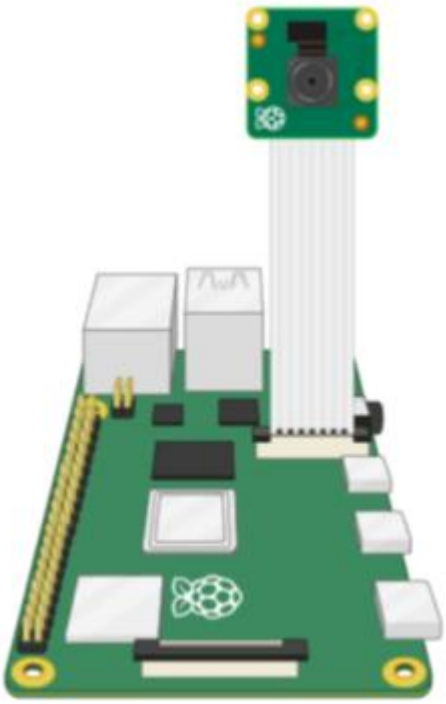
- Sensitivity: Determines how much movement is required to trigger the sensor.
- Time (Delay): Determines how long the sensor remains "triggered" after detecting movement before it resets.

Initial Setup: For this project to work best, use your screwdriver to set the sensitivity potentiometer to its absolute maximum, and set the time potentiometer to its absolute minimum. You can always tweak and vary these settings later if you find the sensor is too sensitive or stays on for too long.

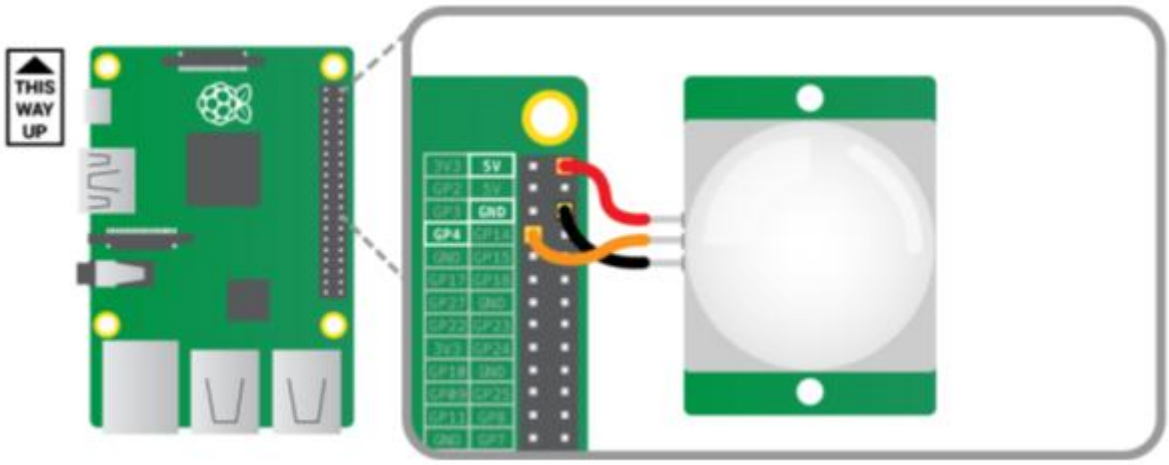
Step 3: Wire the Components Safely

Now it is time to connect your hardware to the Raspberry Pi. Crucial Rule: Always make sure your Raspberry Pi is completely turned off and unplugged before connecting hardware.

1. Connect the Camera: Carefully insert the ribbon cable of the Camera Module into the dedicated camera port on the Raspberry Pi.



2. Connect the PIR Sensor: The PIR sensor needs to send its movement data to the Raspberry Pi. You will connect the data output pin of the PIR sensor directly to the pin labeled GPIO 4 on your Raspberry Pi board. (You will also need to connect the power (VCC) and ground (GND) pins to the respective 5V and GND pins on the Pi).



Step 4: Write the Security Software (Python)

Once your hardware is wired, turn on your Raspberry Pi. Open the programming environment called Thonny, create a new file, and save it immediately as parent_detector.py.

To make our hardware work, we need to import specific libraries. We will use MotionSensor from the gpiozero library to handle the PIR sensor, and the Camera class from the picamzero library to control the Camera Module.

The Logic of the Code:

1. Continuous Monitoring: We use a while True: loop, which is an infinite loop that keeps the program checking for motion forever.

2. Waiting for Action: The program uses pir.wait_for_motion() to pause the code until the sensor detects movement. Once motion is detected, it prints a message to the screen.

3. Recording: The camera begins capturing video using the cam.start_recording() function.

4. Stopping: We don't want to record an empty room forever. The code uses pir.wait_for_no_motion() to wait until the intruder leaves, and then safely stops the video file using cam.stop_recording().

Solving a Critical Bug (The Overwrite Problem): If we simply tell the camera to save the file as intruder.mp4, every time a new person enters the room, the old video will be completely overwritten and deleted. To ensure we keep a video log of everyone (pesky parents or siblings), we need a dynamic filename. We can use Python's time library to automatically find out the current date and time and inject it into the video's filename.

Start with this code and your favourite AI tools to test this code:

Type this into your parent_detector.py file:

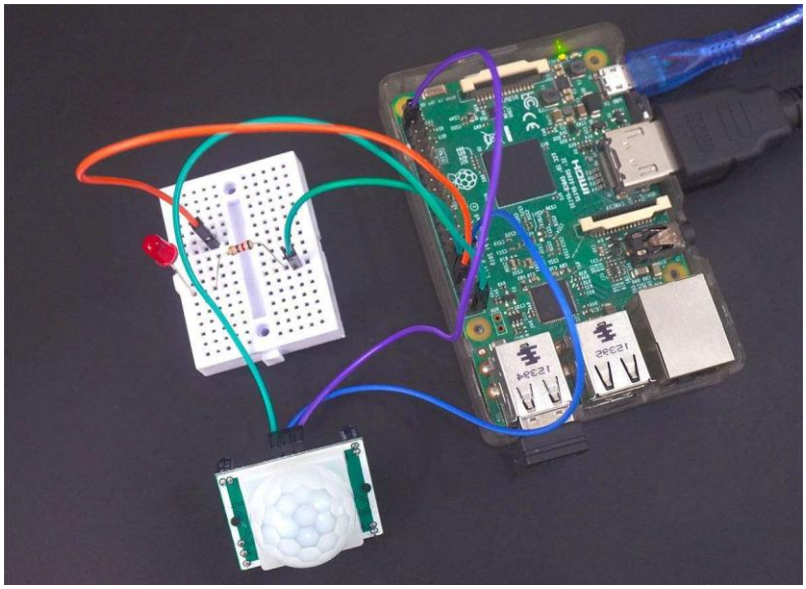
```
from gpiozero import MotionSensor
from picamzero import Camera
import time

# 1. Initialize Hardware
# Set up the PIR sensor on GPIO 4
pir = MotionSensor(4)

# Create a Camera object to control the module
cam = Camera()

print("Security System Armed. Waiting for intruders...")

# 2. Main Security Loop
while True:
    # Program pauses here until movement is detected
    pir.wait_for_motion()
    print("WARNING: Motion detected!")

    # 3. Generate a Unique Filename
    # This creates a string like "20231025-143000" (YearMonthDay-HourMinuteSecond)
    timestamp = time.strftime("%Y%m%d-%H%M%S")
    filename = f"intruder_{timestamp}.mp4"

    # Start recording video evidence to the new file
    print(f"Recording video: {filename}")
    cam.start_recording(filename)

    # 4. Wait for the room to be empty again
    pir.wait_for_no_motion()
    print("Motion stopped.")
```

```
32.
33.     # Stop recording to finalize and save the video file safely
34.     cam.stop_recording()
35.     print("System returning to standby mode.")
```

Now that your code is fully written and your hardware is securely connected, it is time to test if your parent detector actually works in the real world! First, click the Run button inside your Thonny programming environment to start the script. Once the program is running and waiting, deliberately wave your hand directly in front of the PIR motion detector to simulate an intruder walking into the room. Immediately look at your computer screen; you should see the words "Motion detected!" printed out in the console area, confirming that the sensor has successfully triggered the camera to start recording.

After triggering the alarm, you need to test the shut-off mechanism. Step out of the sensor's view, hold completely still, or gently cover the white dome of the sensor with your hand. Wait a few seconds until the console tells you that the motion has stopped and the system is returning to standby mode. This indicates that the program has safely finalized and closed the video file. Finally, open your Raspberry Pi's file manager and navigate to the exact same folder where you saved your *parent_detector.py* script. You should now see a brand new .mp4 video file waiting for you, featuring the unique date and time stamp in its name just as we programmed. Double-click this newly created file to watch the recorded evidence, verify the video quality, and make sure your camera angle captures the doorway perfectly!

Additional pictures:



Source: https://www.electronicwings.com/raspberry-pi/pir-motion-sensor-interfacing-with-raspberry-pi

Source: https://thepihut.com/products/pir-camera-case-for-raspberry-pi-4-3

16.2 YOLO Pose Estimation Recognition

In this guide we will be setting up some with OpenCV and the YOLO pose estimation model family on the Raspberry Pi 5. We will be taking a look at a few of the different YOLO models available, as well as how to optimise them to get smoother FPS', and also how to use the keypoint data generated by the model so you can implement pose estimation into your next project. This has been one of the most fun guides we have made in a while so let's get into it!

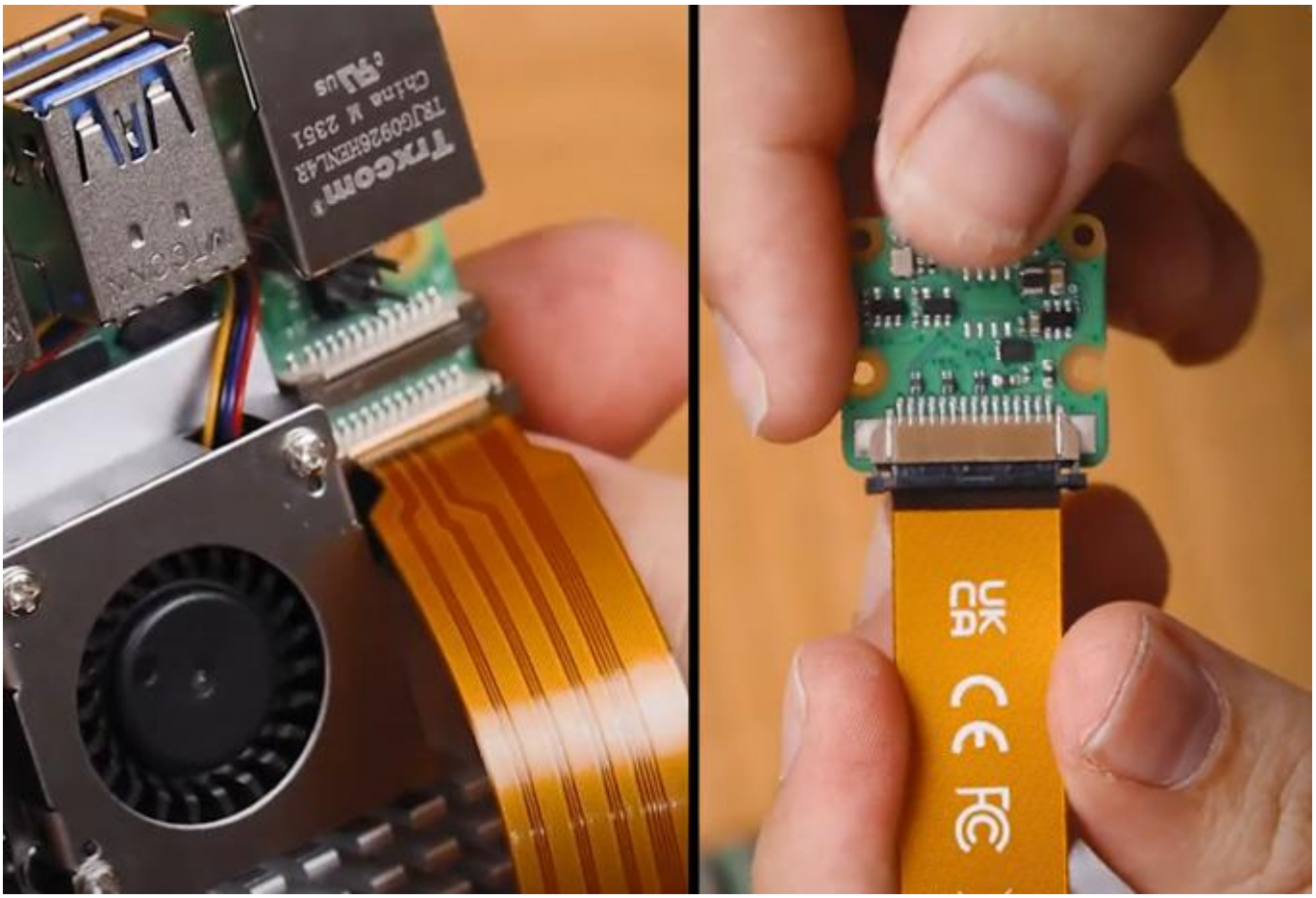
To follow along with this guide you will need a:

- Raspberry Pi 5 - Either a 4GB or 8GB model will work here. Although this could technically be done on a Pi 4, it is far slower than the Pi 5 and would not be a nice experience, and for those reasons, we haven't tested on a Pi 4
- Pi Camera - We are using the Camera Module V3
- Adapter Cable - The Pi 5 comes with a different-sized CSI camera cable and your camera may come with the older thicker one so it's worth double-checking. The Camera Module V3 WILL need one
- Cooling Solution - We are using the active cooler (computer vision will really push your Pi to its limits)
- Power Supply
- Micro SD Card - At least 16GB in size
- Monitor and Micro-HDMI to HDMI Cable
- Mouse and Keyboard

Hardware Assembly

In terms of hardware assembly, it's pretty light here. Connect the thicker side of the cable to the camera, and the thinner side to the Pi 5. These connectors have a tab on them - lift them up, then insert the cable into the slot. Once it is sitting in there nice and square, push the tab back down to clamp the cable into place.

Just keep an eye out as these connectors only work in one orientation, and they can be fragile so avoid bending them tightly (a little bit is okay).



Installing Pi OS

First things first, we need to install Pi OS onto the micro SD card. Using the **Raspberry Pi Imager**, select Raspberry PI 5 as the Device, Raspberry Pi OS (64-bit) as the Operating system, and your microSD card as the storage device.

Same procedure as for PiRacer.

Setting up a Virtual Environment and Installing Libraries

With the introduction of Bookworm OS in 2023, we are now required to use Virtual Environments (or venv), as they are an isolated space on the Pi where we can experiment without the risk of harming the rest of our Pi OS or projects. We have all the needed commands and instructions in this guide.

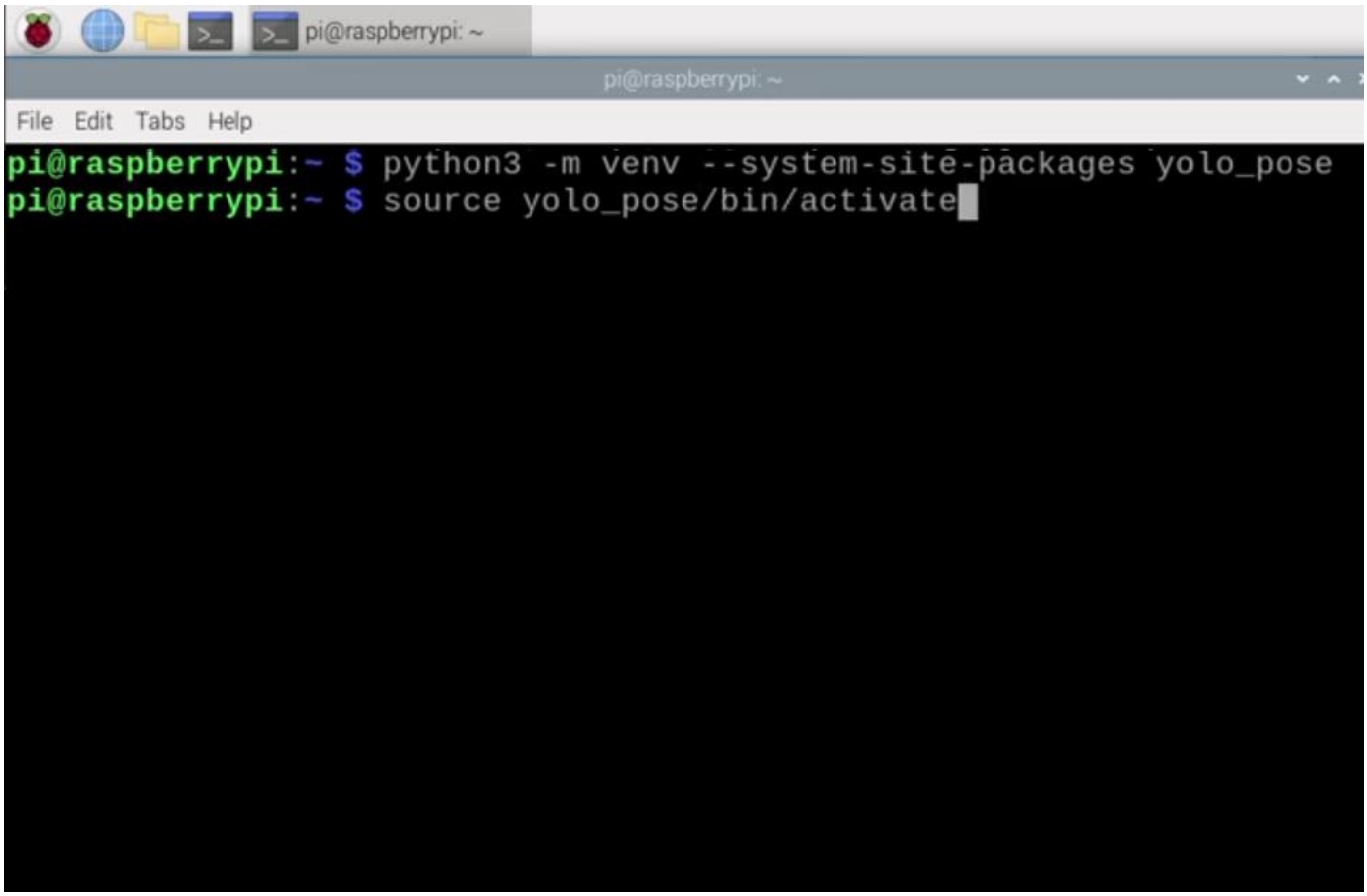
To create a virtual environment, open a new terminal window and type in:

python3 -m venv --system-site-packages yolo_pose

After creating the venv, we can enter into it by typing in:

source yolo_pose/bin/activate

After doing so you will see the name on the virtual environment to the left of the green text - this means we are correctly working within it. If you ever need to re-enter this environment (for example if you close the terminal window you will exit the environment), just type in the source command above again.



Now that we are working in a virtual environment, we can start installing the required packages. First, ensure that PIP (the Python package manager) is up to date by entering the three following lines:

sudo apt update

sudo apt install python3-pip -y

pip install -U pip

Then install the Ultralytics Package with:

pip install ultralytics[export]

The lovely folks at Ultralytics have been one of the key developers and maintainers of the newest YOLO models. This package of theirs is going to do much of the heavy lifting and will install OpenCV as well as all the required infrastructure for us to run YOLO.

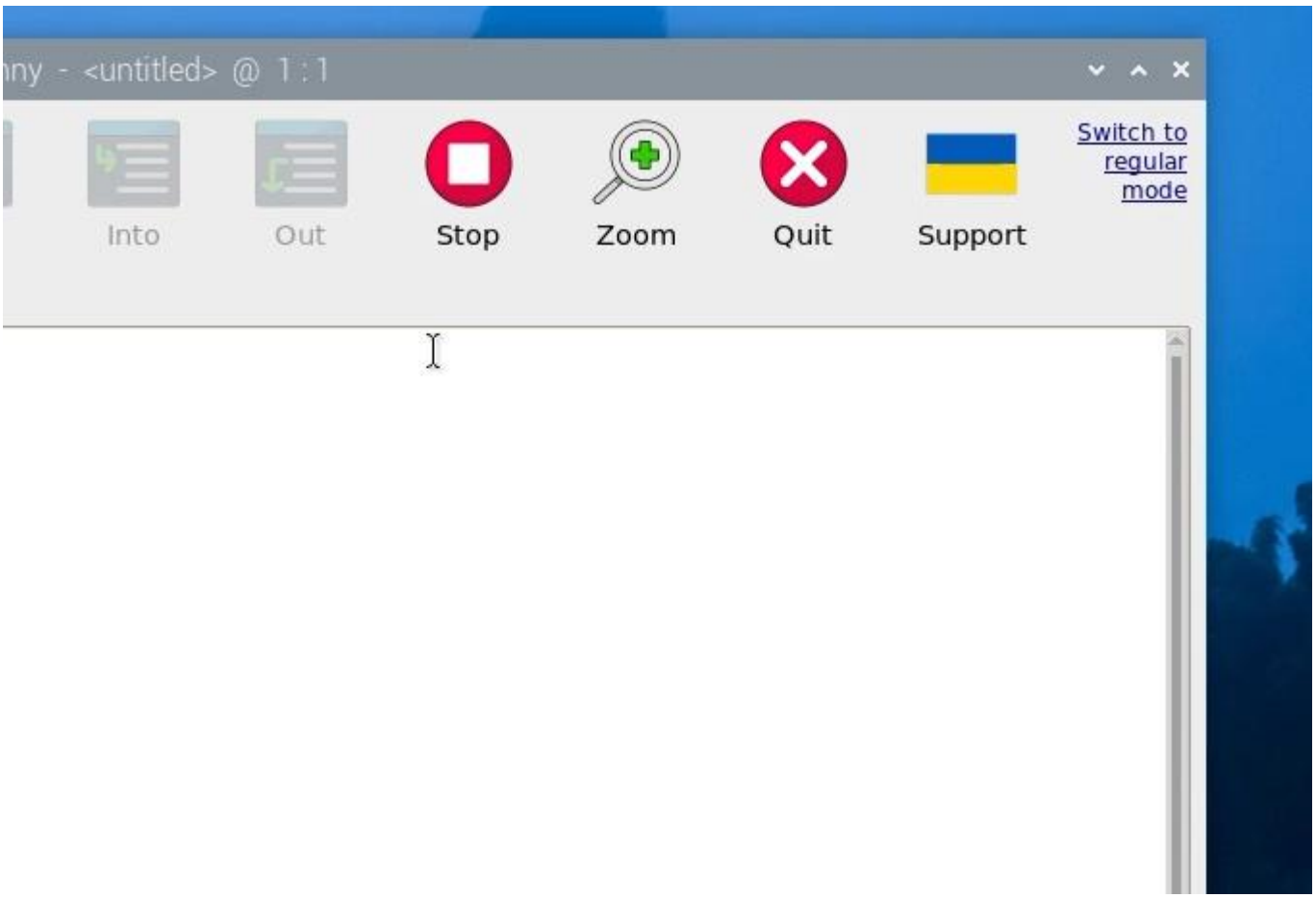
This process will also install quite a large amount of other packages, and as a result, is prone to failing. If your installation fails (it will display a whole wall of red text), just type in the Ultralytics install line again and it should resume. In rare cases, the install line may need to be repeated a few times.

Once that has finished installing, reboot the Raspberry Pi. If you want to be a power user, you can do so by typing into the shell:

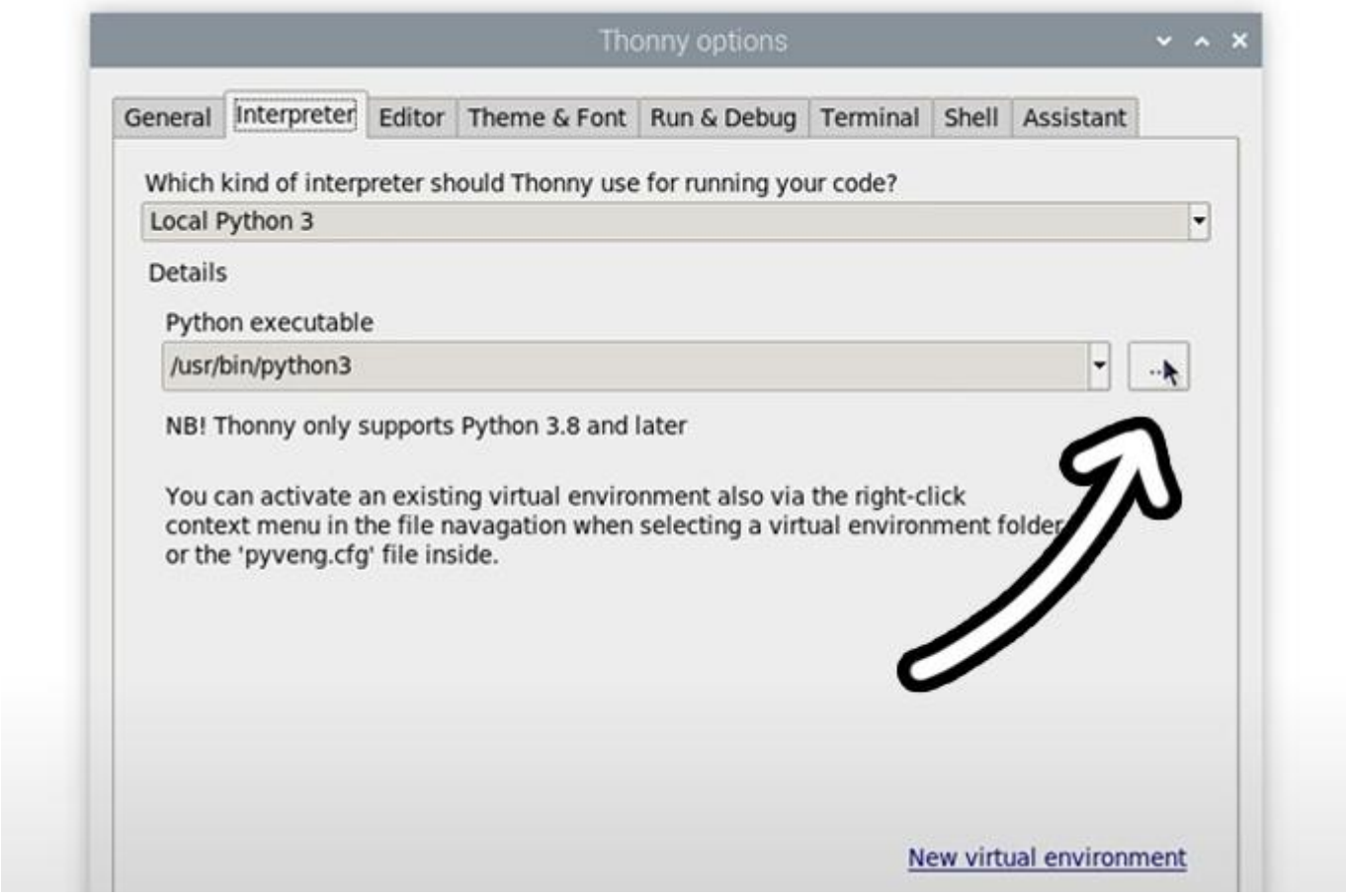
reboot

We have one more thing to do, and that is to set up Thonny to use the virtual environment we just created. Thonny is the program we will be running all of our code out of and we need to get it to work out of the same venv so that it has access to the libraries we installed.

The first time you open Thonny it may be in the simplified mode, and you will see a "switch to regular mode" in the top right. If this is present click it and restart Thonny by closing it.



Now enter the interpreter options menu by selecting Run > Configure Interpreter. Under the Python executable option, there is a button with 3 dots. Select it and navigate to the Python executable in the virtual environment we just created.



This will be located under home/pi/yolo_pose/bin and in this file, you will need to select the file called "python3". Hit okay and you will now be working in this venv.

Whenever you open Thonny, it will now automatically work out of this environment. You can change the environment you are working out of by selecting it from the drop-down menu under the Python executable in the same interpreter options menu. If you wish to exit the virtual environment, select the option bin/python3.

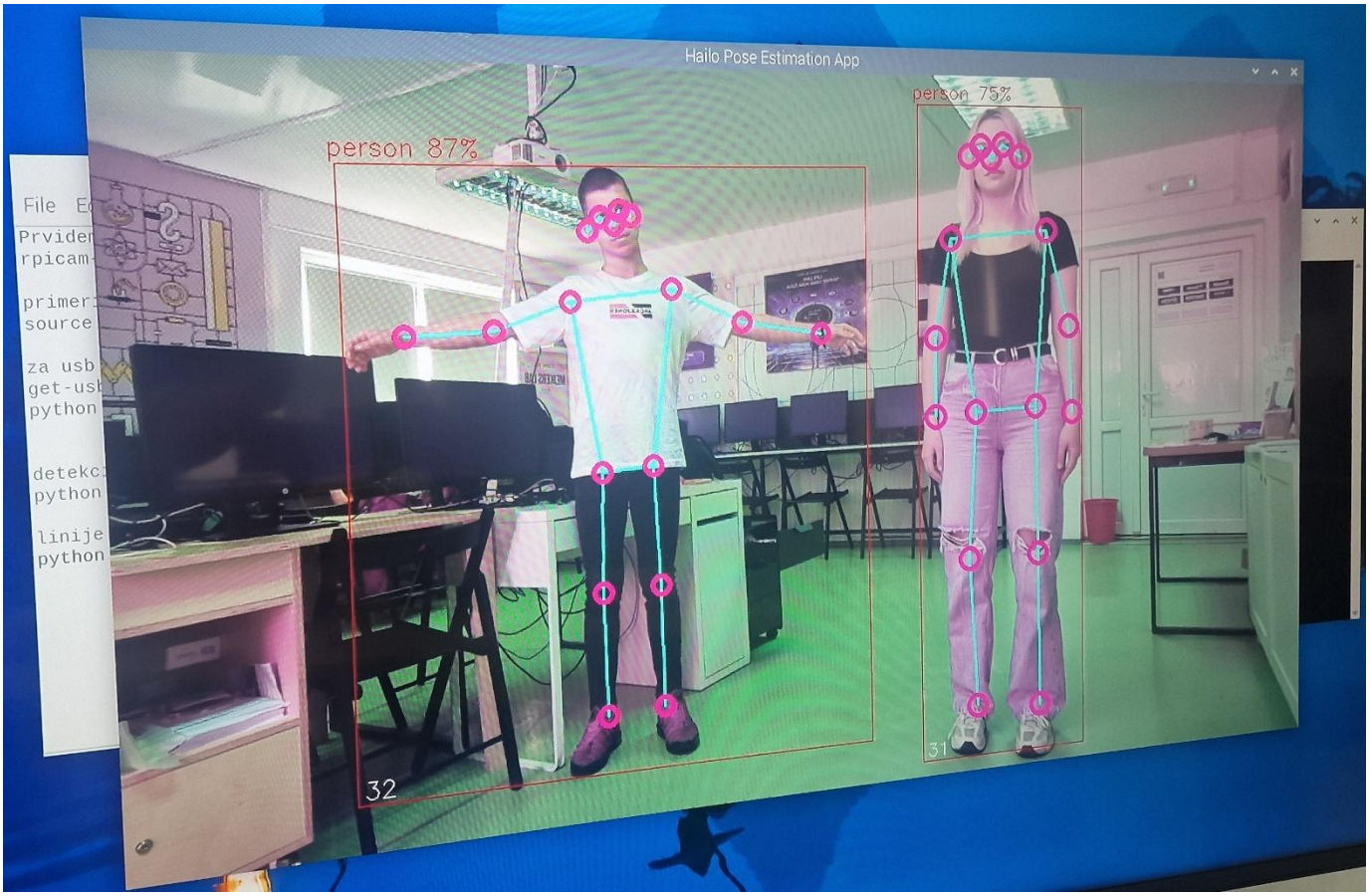
Running Pose Estimation

Now that we have our libraries installed and Thonny is working out of the virtual environment, we can run our pose estimation script. Go ahead and extract the **project zip folder**(download from our server) to a convenient place like the desktop. In there, you will find the first script we will use *"pose demo.py".* Open it Thonny and hit the big green run button. The first time you run this it may install a few extra needed things (all automatically), and after a few seconds you should see a preview window appear with your pose estimation running.

A few things should be happening here. First YOLO will be trying to detect humans, and if it recognises one it will draw a box around it with the confidence rating at the top. The important thing is that it will be placing points at where it thinks some essential places of your body are (these are called keypoints), and it will be

drawing lines between these points to estimate the pose and orientation of the person. In the top right will also be the FPS this is running at (which we will improve in a bit).

And that's it! With these few steps, we already have our Pi running pose estimation!

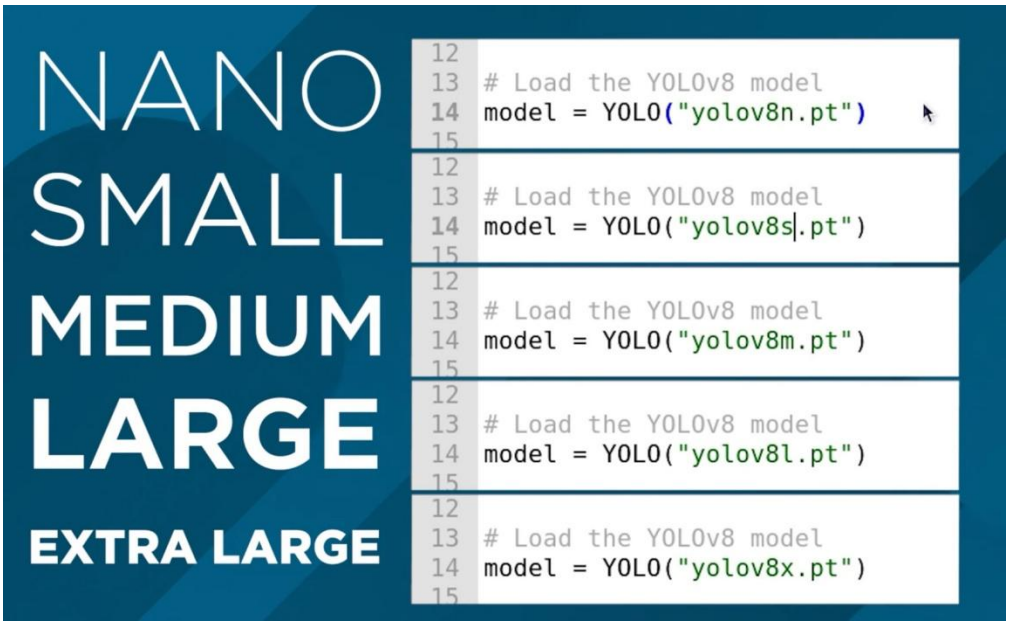


Changing YOLO Models

So far we have been running YOLO11, and one of the beauties of this Ultralytics package is that we can simply swap out a single line in the code to completely change the model. We can use this to run a more advanced YOLO11 model, or even an older model. All you need to change is this line here in the setup:

```
# Load our YOLO11 model
model = YOLO("yolo11n-pose.pt")
```

This line is currently using the nano model which is the smallest, least powerful, but fastest model of YOLO11, and we can change this line to run one of the different sizes that this model comes in by changing the single letter after "11" as shown on the right. If you change this line to another model size and run it, the script will automatically download the new model (which can be in the 100s of Mb for the larger models).



The difference between these models is a trade-off between pose-estimation performance and FPS. The larger the model, the better it is at estimating the parts of your body that might not be seen by the camera, as well as more complex angles, and frames with more people in it, however, you can expect to only get 1 frame processed every 10 seconds! We will increase this in the next step.

The nano model on the other hand runs the fastest, getting about 1.5 FPS without optimisation, but it doesn't have the processing power of the larger models. For pose estimation, you can get away with the nano model most of the time as it is usually good enough for your needs, but if you need something a bit more powerful, keep increasing the model size to fit your needs.

In this line, we can also change the version of YOLO running. You can revert to an older model if you want, or you can utilise a newer model. This guide will eventually be outdated and if Ultralytics releases YOLO13, you should simply be able to change the line to the following to start using the newer YOLO version:

```
# Load our YOLO11 model
model = YOLO("yolo13n-pose.pt")
```

Increasing Processing Speed

There are 2 things we can do to increase FPS on the Pi and the most effective way is to convert the model to a format called NCNN. This is a model format more optimised to run on ARM-based processors like the Raspberry Pi's. Open up the script called "ncnn conversion.py" and you will find the following:

```
from ultralytics import YOLO

# Load a YOLO11n PyTorch model
model = YOLO("yolo11n-pose.pt")

# Export the model to NCNN format
model.export(format="ncnn", imgsz=640)  # creates 'yolov11n-pose_ncnn_model'
```

To use this script, first specify the model you wish to convert. This uses the same naming conventions we talked about in the last section. Then the model format "ncnn" is specified as the output format, as well as the

resolution. For now keep this at the default of 640. The first time you run this script it will download some more additional things it needs, but it should only take a few seconds to run the actual conversion.

Once that has finished, in the folder the scripts live in your will find a new folder called something along the lines of "yolo11n-pose_ncnn_model". Copy the name of this file and return to our demo script from earlier.

You now need to tell the script to use this model that we created by changing the model line to the name of that folder it just created. It should look something like this:

```
# Load our YOLO11 model
model = YOLO("yolo11n-pose_ncnn_model")
```

17. REFERENCES

Bradski, G. (2000). The OpenCV library. *Dr. Dobb's Journal of Software Tools*, 25(11), 120–125.

Cabello, R., & Three.js Authors. (2023). *Three.js JavaScript 3D library* (Version 0.157.0) [Computer software]. https://threejs.org/

Dobot Robotics. (2024). *Dobot Magician user guide & DobotStudio operating instructions*. Shenzhen Center Stage Co., Ltd. https://www.dobot-robots.com/

DonkeyCar Open Source Community. (n.d.). *Donkey Car: An open source DIY self driving platform for small scale cars*. https://docs.donkeycar.com/

Fraser, N. (2013). *Google Blockly: A visual programming framework for web and mobile apps*. Google Developers. https://developers.google.com/blockly

Jocher, G., & Qiu, J. (2024). *Ultralytics YOLO11: Real-time object and pose estimation* [Computer software]. GitHub. https://github.com/ultralytics/ultralytics

Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. *In European Conference on Computer Vision*, 740–755. https://doi.org/10.1007/978-3-319-10602-1_48

Raspberry Pi Foundation. (2023). *Raspberry Pi 5 documentation and getting started guide*. https://www.raspberrypi.com/documentation/

Waveshare Wiki. (n.d.). *PiRacer AI Kit: AI automated driving robot powered by Raspberry Pi*. https://www.waveshare.com/wiki/PiRacer_AI_Kit

World Wide Web Consortium. (2020). *WebXR Device API* [W3C Candidate Recommendation]. https://www.w3.org/TR/webxr/